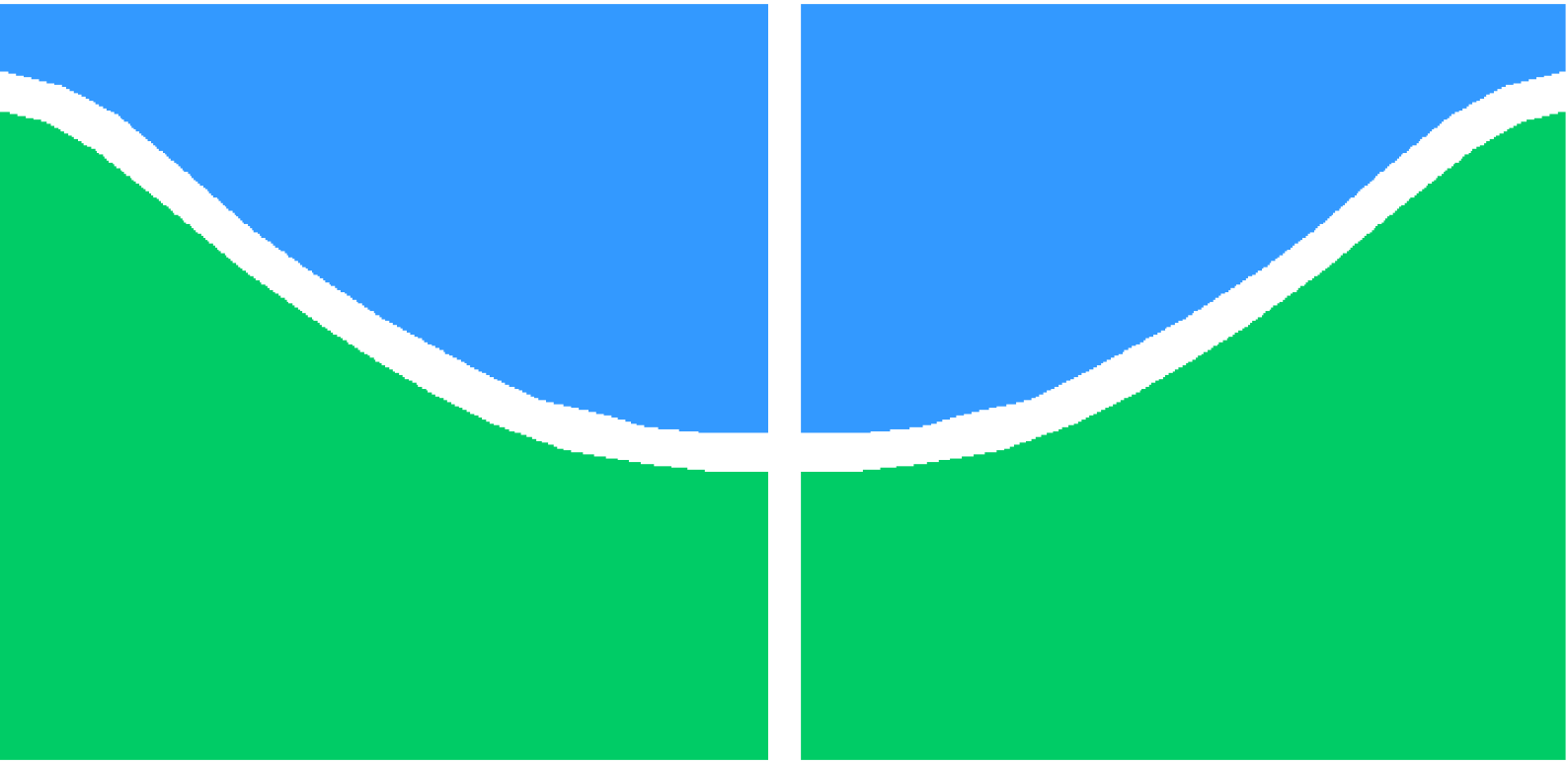


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia Eletrônica

Desenvolvimento de um sistema para auxílio à locomoção de deficientes visuais através da implementação em arquiteturas reconfiguráveis da transformada Census para estimação de distância usando visão estéreo

Autor: Everaldo Henrique Diniz

Felipe Medeiros Demarchi

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília, DF

2013

Everaldo Henrique Diniz e Felipe Medeiros Demarchi

Desenvolvimento de um sistema para auxílio à locomoção de deficientes visuais através da implementação em arquiteturas reconfiguráveis da transformada Census para estimação de distância usando visão estéreo

Monografia submetida ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília, DF

2013

Everaldo Henrique Diniz e Felipe Medeiros Demarchi

Desenvolvimento de um sistema para auxílio à locomoção de deficientes visuais através da implementação em arquiteturas reconfiguráveis da transformada Census para estimação de distância usando visão estéreo/ Everaldo Henrique Diniz e Felipe Medeiros Demarchi. – Brasília, DF, 2013-

82 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2013.

1. Transformada Census. 2. FPGA. I. Prof. Dr. Daniel Mauricio Muñoz Arboleda. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Desenvolvimento de um sistema para auxílio à locomoção de deficientes visuais através da implementação em arquiteturas reconfiguráveis da transformada Census para estimação de distância usando visão estéreo

CDU 00:000:000.0

Everaldo Henrique Diniz e Felipe Medeiros Demarchi

Desenvolvimento de um sistema para auxílio à locomoção de deficientes visuais através da implementação em arquiteturas reconfiguráveis da transformada Census para estimação de distância usando visão estéreo

Monografia submetida ao curso de graduação em (Engenharia Eletrônica) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia Eletrônica).

Trabalho aprovado. Brasília, DF, 12 de Dezembro de 2013:

Prof. Dr. Daniel Mauricio Muñoz Arboleda
Orientador

Prof. Dr. Cristiano Jacques Miosso
Convidado 1

Prof. MSc. Gerardo Idrobo Pizo
Convidado 2

Brasília, DF
2013

Resumo

Este trabalho propõe um sistema de auxílio para deficientes visuais no intuito de aumentar a independência e gerar uma melhor qualidade de vida. O sistema está baseado no cálculo de correspondências e disparidades entre duas imagens estereoscópicas, para o qual foi usada a transformada Census e o cálculo da distância de Hamming visando estimar a distância frontal até os obstáculos. O sistema proposto está composto por um par de câmeras e um dispositivo FPGA (*Field Programmable Gate Array*) que acelera a execução dos algoritmos envolvidos. Uma ferramenta de geração automática de código VHDL foi construída no intuito de acelerar o tempo de desenvolvimento da implementação das arquiteturas de hardware para diferentes tamanhos de imagem usando máscaras de 3x3, 5x5, 7x7, 9x9 e 11x11 pixels. Todas as arquiteturas foram sintetizadas e um estudo de escalabilidade em termos de consumo de recursos foi realizado. Os resultados de síntese demonstram que as arquiteturas de *hardware* são eficientemente mapeadas em dispositivos FPGA comerciais, alcançando uma frequência de operação de 180MHz aproximadamente. Duas memorias ROM, uma para cada imagem, foram instanciadas visando emular o fluxo de pixels das câmeras esquerda e direita, e simulações comportamentais foram realizadas no intuito de verificar o comportamento lógico das arquiteturas. A mesma técnica foi implementada em *hardware* e *software* e a comparação numérica entre as implementações demonstram a eficiência das arquiteturas propostas, além disso, é possível concluir que as arquiteturas propostas apresentam resultados eficientes em termos da qualidade do mapa de disparidade. Um fator de aceleração de 211 vezes foi alcançado para o cálculo do mapa de disparidade se comparado com uma implementação em *software* usando um *Desktop* convencional Intel Core i7 operando a 3.4 GHz. É importante ressaltar que a utilização da transformada Census básica acrescenta ruído no processo de cálculo de correspondência entre as imagens e que a utilização das transformadas modificadas melhoram a performance dos resultados.

Palavras-chaves: FPGA, visão estereoscópica, transformada Census, distância de Hamming, sistemas embarcados, tecnologias assistivas.

Abstract

This work proposes a system to help visually impaired people in order to improve their independence and quality of life. The system is based on the disparity map computation between two stereoscopic images and uses the Census transform and the Hamming distance for estimating the distance between the system and the obstacles. The proposed system is composed of a stereoscopic system and a FPGA (Field Programmable Gate Array) which accelerates the execution time of the involved algorithms. In this work a VHDL code generator was created for implementing the hardware architectures for different images sizes and mask sizes of 3x3, 5x5, 7x7 and 9x9 pixels. All the hardware architectures were synthesized and a scalability analysis in terms of hardware resources consumption was provided. Synthesis results demonstrates that the hardware architectures are efficiently mapped on commercial FPGA devices, achieving an operational frequency of 180MHz approximately. Two ROM memories, one for each image, were instantiated in order to emulate the stream of pixels from the left and right cameras and behavioural simulations were performed in order to verify the logic implementation of the architectures. Numerical comparisons between hardware and software implementations demonstrated the effectiveness of the proposed architectures. A speed up factor of 211 times was achieved for computing the disparity map between two images if compared with a software implementation using a Desktop solution Intel Core i7 operating at 3.4 GHz.

Key-words: FPGA, stereoscopic vision, Census transform, Hamming distance, embedded systems, assistance technologies.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ASIC	*Application Specific Integrated Circuits*
DSPs	*Digital Signal Processing*
FPGA	*Field Programmable Gate Array*
FF	Flip-Flops
HDLs	*Hardware Description Language*
LUTs	*Look Up Tables*
OS	Sistema Operacional
PLD	*Programmable Logic Devices*
RAM	*Random Access Memory*
ROM	*Read Only Memory*
SAD	Soma das Diferenças Absolutas
SSD	Soma dos Quadrados das Diferenças
TC	Transformada Census
TR	Transformada Rank
VHDL	*Very High Speed Integrated Circuits Hardware Description Language*
VLSI	*Very Large Scale Integration*

Sumário

1 Introdução

Entre todos os sentidos do corpo humano a visão é o que gera maior independência a um indivíduo. Segundo a Classificação Internacional de Deficiências, Incapacidades e Desvantagens um deficiente visual é aquele com a perda total ou parcial, congênita ou adquirida, da visão (MANUAL. . . , 2006). De acordo com o censo do Instituto Brasileiro de Geografia e Estatística (IBGE) de 2010 o Brasil possui mais de 6,5 milhões de deficientes visuais e a cada 5 segundos uma pessoa se torna cega no mundo.

O presente trabalho está inserido no desenvolvimento de um projeto de pesquisa mais amplo, proposto e coordenado pelo professor Daniel Muñoz, que tem como principal objetivo desenvolver um protótipo funcional de um dispositivo que permita o deslocamento de deficientes visuais com cegueira parcial ou total em recintos fechados, com consequente melhoria da qualidade de vida pelo aumento da autonomia na locomoção. No dispositivo almejado são considerados diversos sensores de detecção de distância, tais como, sensores de ultrassom e infravermelho, assim como, sistemas de visão estéreo e informação proveniente de acelerômetros e giroscópios (vide figura 1). Adicionalmente, serão realizados processos de fusão sensorial e filtragem visando melhorar a estimativa das variáveis. O sistema possui algoritmos de localização e realiza um mapa de distância aos obstáculos no entorno e, através de uma interface sonora e eletromecânica, baseada em dispositivos piezoelétricos, indica ao usuário a direção em que deve caminhar no intuito de evadir obstáculos.

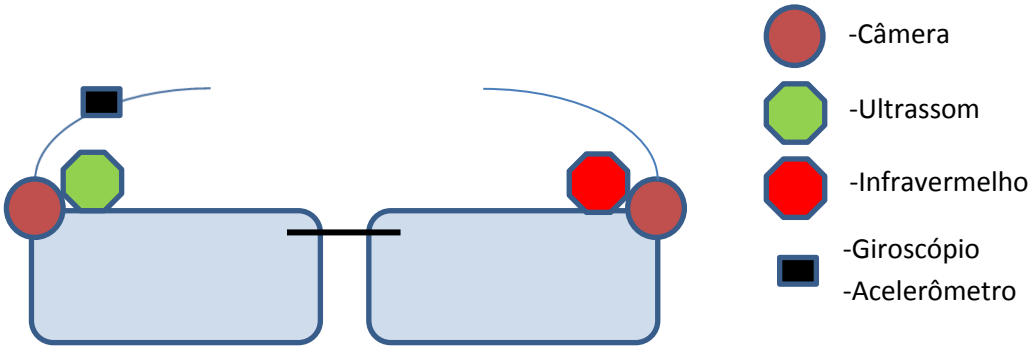


Figura 1 – Componentes e sensores do dispositivo de auxílio a deficientes visuais.

Sistemas de visão estéreo usam a informação de duas imagens do mesmo local através de dois pontos de vista distintos. Com esse tipo de sistemas é possível extrair informações do espaço tridimensional, tais como distância, tamanho e posição. As imagens podem ser capturadas por duas ou mais câmeras ou por uma câmera em movimento. No

entanto, com a câmera em movimento o intervalo de tempo entre as duas capturas não pode ser elevado, pois o ambiente está sujeito a modificações (BAILEY, 2011).

1.1 Descrição do Problema

Os algoritmos usados em sistemas de visão estéreo comumente estão baseados em transformações por correspondência ou por área, as quais requerem um processamento de dados intensivo e contínuo. De forma geral, esses algoritmos possuem um tempo de execução elevado devido ao seu alto custo computacional. Este problema torna-se crítico quando os algoritmos são aplicados em soluções embarcadas, em que geralmente são usados microprocessadores que trabalham com uma baixa frequência de relógio se comparado com processadores de propósito geral (SASS, 2010). Contudo, deve ser ressaltado que os algoritmos de processamento para cálculo de correspondências entre imagens estereoscópicas possuem paralelismo intrínseco que pode ser explorado no intuito de melhorar o desempenho em termos do tempo de execução e da qualidade do resultado.

Dispositivos FPGAs (*Field Programmable Gate Array*) são uma solução adequada para processamento espacial de imagens, permitindo implementar arquiteturas de *hardware* dedicadas para o processamento paralelo. Adicionalmente, devido a que as soluções em FPGA geralmente operam a baixas frequências de relógio, facilita a sua aplicação em sistemas embarcados em que o consumo de energia é um fator ponderante. Uma vantagem adicional do desenvolvimento de arquiteturas de *hardware* dedicadas baseada em FPGAs é a possibilidade de fabricação de circuitos integrados de aplicação específica (ASICs) visando explorar as capacidades de processamento de alto desempenho e baixo consumo de energia dos sistemas embarcados desenvolvidos (MEYER-BAESE, 2004).

Uma revisão do estado da arte demonstrou que poucos estudos têm sido realizados para a implementação de arquiteturas dedicadas para o aceleramento de algoritmos de cálculo de correspondências entre imagens estereoscópicas. A revisão bibliográfica mostrou que são poucas as soluções baseadas no cálculo de correspondências entre imagens para o caso específico de soluções embarcadas de alto desempenho focadas no auxílio à locomoção de deficientes visuais.

É com base na análise das técnicas de cálculo de correspondências ente imagens estereoscópicas e com foco em soluções simples que proporcionem melhorias de desempenho em termos do tempo de execução que o presente trabalho tem sua motivação.

1.2 Aspectos Metodológicos

O trabalho foi desenvolvido seguindo a metodologia tradicional para projeto de circuitos digitiais em FPGAs. As arquiteturas de *hardware* foram descritas usando a lin-

guagem VHDL (*Very High Speed Integrated Circuits Hardware Description Language*). Para efeitos de validação das arquiteturas, foram realizadas simulações comportamentais, assim como comparações numéricas entre as implementações de *hardware* e *software*. As implementações de *software* foram realizadas usando a linguagem C e executadas em um processadores de escritório Intel Core i7 operando a 3.4 GHz, 8GB RAM, Windows OS.

Assim, a metodologia de desenvolvimento segue as seguintes fases:

Na fase 1 foi realizada uma análise de desempenho em termos do tempo de execução (análise de *profile*) da implementação *software* visando identificar as funções com maior custo computacional no intuito de realizar a sua implementação em *hardware*, viabilizando assim o cálculo do mapa de disparidade entre as imagens em tempo real.

Na fase 2 foram realizadas as implementações de *hardware* usando uma arquitetura modular com metodologia *bottom-up*, facilitando a verificação do circuito proposto. É importante salientar que na arquitetura proposta foram usadas duas memórias ROM, uma para cada imagem, visando emular o fluxo de pixels das câmeras esquerda e direita. Isto possibilita a verificação do circuito sem o uso direto do sistema estereoscópico e sua correspondente calibração.

Na fase 3 foi desenvolvida uma ferramenta de geração automática de código VHDL capaz de gerar códigos para diferentes tamanhos de imagens e para diferentes tamanhos de máscaras de pixels. Esta ferramenta permitiu acelerar o tempo de desenvolvimento das implementações. Assim, as arquiteturas para máscaras 3x3, 5x5, 7x7 e 9x9 foram implementadas.

Na fase 4 os resultados de síntese lógica, consumo de recursos e frequência de operação dos circuitos para diferentes tamanhos de máscara foram coletados. Adicionalmente, foram realizadas simulações comportamentais no intuito de verificar o comportamento lógico das arquiteturas. Os resultados foram comparados com uma implementação em *software* e os mapas de disparidade foram comparados usando uma métrica baseada no erro quadrático médio (MSE) e o tempo de execução entre as implementações *hardware* e *software* foi estimado visando calcular o fator de aceleração. Finalmente, foi realizada uma implementação real em um dispositivo FPGA Startan6 da Xilinx usando uma máscara de 5x5 verificando-se a eficiência dos circuitos desenvolvidos.

1.3 Objetivos

O objetivo geral do presente trabalho é desenvolver uma arquitetura de *hardware* dedicada para o cálculo do mapa de disparidade entre duas imagens adquiridas através de um sistema de visão estéreo de forma que permita calcular a distância frontal entre o deficiente visual e os obstáculos.

Objetivos específicos

- Implementação em *software* usando código C, validação e análise de desempenho do algoritmo de cálculo de disparidade entre duas imagens.
- Desenvolvimento, simulação e validação da implementação em FPGAs da transformada Census para cálculo de correspondências entre duas imagens.
- Desenvolvimento, simulação e validação da implementação em FPGAs do cálculo da distância de Hamming para estimação da disparidade entre duas imagens.

1.4 Contribuições do Trabalho

As principais contribuições do trabalho são as seguintes:

- Desenvolvimento de uma arquitetura de *software* para conversão da base binária para decimal.
- Desenvolvimento de uma arquitetura de *software* dedicada para a estimação do cálculo do mapa de disparidade entre duas imagens estereoscópicas, baseada na transformada Census e no cálculo da distância de Hamming, permitindo estimar a distância entre um deficiente visual e os obstáculos no ambiente.
- Desenvolvimento de uma arquitetura de *hardware* dedicada para a estimação do cálculo do mapa de disparidade entre duas imagens estereoscópicas, baseada na transformada Census e no cálculo da distância de Hamming, permitindo estimar a distância entre um deficiente visual e os obstáculos no ambiente.
- Desenvolvimento de uma ferramenta de geração automática de código VHDL que permite a implementação da arquitetura proposta para diferentes tamanhos de imagem e de máscara de pixels.

1.5 Organização do Trabalho

O presente documento está organizado da seguinte maneira. A Seção 2 apresenta os objetivos do trabalho. A Seção 3 apresenta a fundamentação teórica dos algoritmos envolvidos no processamento de imagens estereoscópicas. A Seção 4 apresenta as implementações realizadas. A Seção 5 apresenta os resultados de síntese, execução e simulação. A Seção 6 apresenta uma discussão dos resultados obtidos. Finalmente, a Seção 7 apresenta as conclusões do trabalho e propostas para trabalhos futuros.

2 Fundamentação Teórica

2.1 Processamento Digital de Imagens

Segundo Gonzalez *et al.* (GONZALES, 2009), uma imagem pode ser definida como uma função bidimensional f(x,y) em que as variáveis x e y denotam coordenadas espaciais e f no par de coordenadas (x,y) é um valor que representa a amplitude, intensidade ou escala de cinza da imagem. Adicionalmente, o termo processamento digital de imagens é definido como o processamento de imagens através de computadores digitais. Este tipo de processamento envolve processos de extração de características, reconhecimento de padrões mediante o uso de modelos matemáticos que permitem manipular as imagens no intuito de formar uma base de conhecimento formal da cena que está sendo representada.

Os passos fundamentais para o processamento digital de imagens são apresentados na Figura 2 (GONZALES, 2002). Esta estrutura de processamento pode-se dividir em três passos fundamentais: aquisição, pré-processamento e processamento.

A aquisição pode ser feita de várias formas, tanto a captura por diversos sensores ou até mesmo a aquisição de uma imagem previamente digitalizada. É importante ressaltar que uma imagem não necessariamente é adquirida por sensores que captam a luz no espectro visível (longitude de onda de 390 nm a 700 nm), senão também imagens adquiridas no espectro de raios X, raios gama ou imagens térmicas (infravermelho).

No intuito de preparar a imagem para a etapa de processamento comumente é realizado um pré-processamento que envolve tarefas de transformação de cores, tratamento de ruído, ajuste do tempo de abertura, ajuste de escala, entre outros.

O processamento da imagem é a etapa onde são executados algoritmos para extração de características de interesse para a aplicação. Neste processamento enquadram-se técnicas como transformadas no domínio da frequência, transformadas no domínio espacial, detecção de bordas, detecção de cantos, filtragem, interpolação, segmentação, correlação, processamento morfológico, reconhecimento de padrões, entre outras.

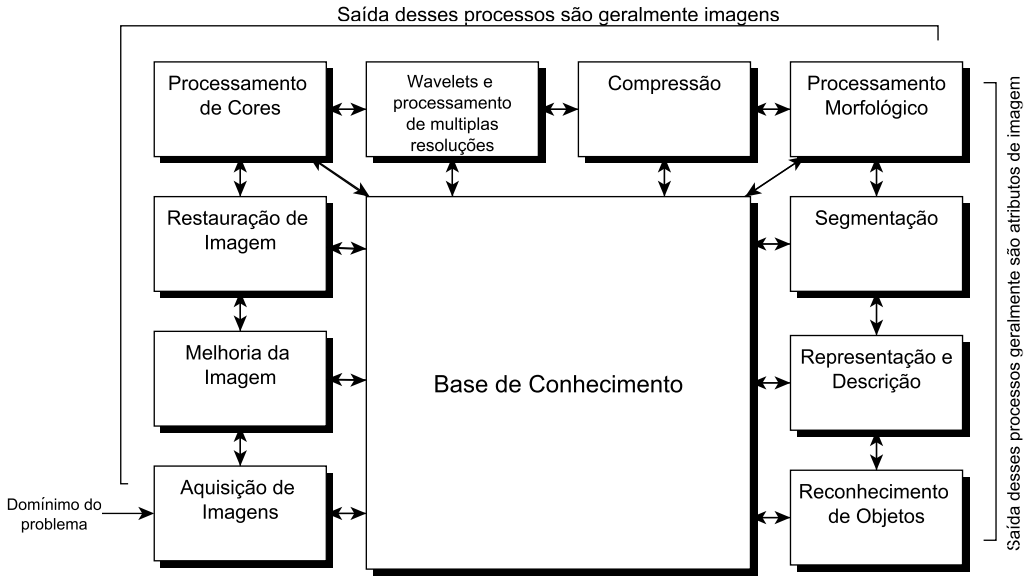


Figura 2 – Passos fundamentais no processamento digital de imagens. (Modificado (GONZALES, 2002).

2.2 Sistema de Visão Estéreo

Um sistema de visão estéreo captura duas imagens do mesmo local através de dois pontos de vista distintos. Com esse tipo de sistema é possível extrair várias informações do espaço tridimensional, como, distância, tamanho e posição.

Para obter essas informações é necessário:

- Conhecer a geometria do sistema;
- Capturar as imagens;
- Tratar e processar as imagens;
- Interpretar a informação.

A geometria leva em consideração as características, configurações e posições das câmeras utilizadas. As imagens podem ser capturadas por duas ou mais câmeras ou por uma câmera em movimento. No entanto, com a câmera em movimento o intervalo de tempo entre as duas capturas não pode ser elevado, pois o ambiente está sujeito a modificações. No caso específico de um sistema de visão estéreo em que as câmeras têm a mesma distância focal, é possível, através de um processamento baseado em correspondência entre imagens, calcular a disparidade no intuito de estimar a distância entre objetos e o sistema observador.

2.2.1 Geometria do sistema

Neste trabalho foi escolhido um sistema com duas câmeras do mesmo fabricante (mesma distância focal) e fixadas paralelamente entre si, ou seja, possuem todos os eixos horizontais na mesma posição. Este tipo de configuração pode ser explicado pelo modelo apresentado por Esteves. (ESTEVES, 2012) o qual está representado na Figura 3. É possível observar que um mesmo ponto P encontra-se em posições diferentes em cada imagem, porém, possui a mesma coordenada Y. A distância horizontal entre essas posições chama-se disparidade. No sistema apresentado, com câmeras idênticas e bem alinhadas, as imagens da esquerda e da direita definem uma reta paralela ao eixo X denominada linha epipolar.

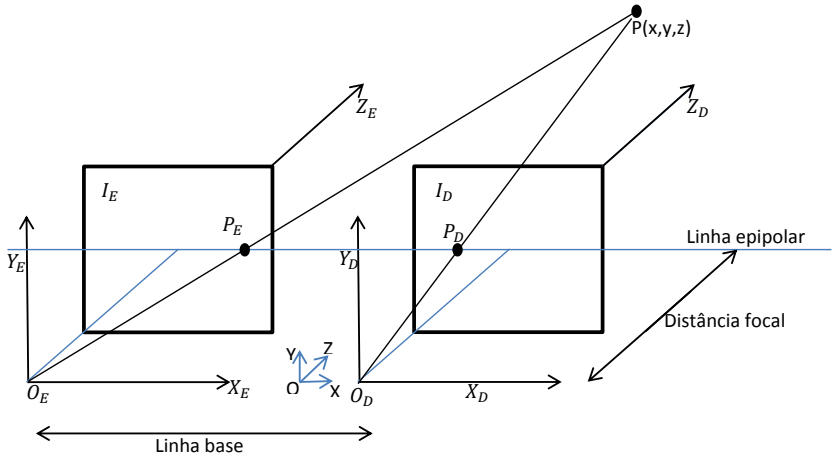


Figura 3 – Geometria do sistema para formação de imagens estereoscópicas (MARTÍNEZ, 2010)

Para facilitar a análise do sistema de visão estéreo pode-se simplificar o modelo da Figura 3 para o sistema mostrado na Figura 4.

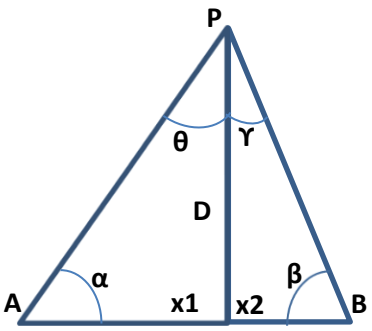


Figura 4 – Simplificação da geometria para formação de imagens estereoscópicas (ESTEVES, 2012).

Com a utilização de trigonometria obtém-se o valor da distância D (distância do ponto P ao sistema observador). A partir da lei dos senos tem-se

$$\frac{D}{\sin\beta} = \frac{x2}{\sin\gamma} \rightarrow X2 = \frac{D\ \sin\gamma}{\sin\beta} \tag{2.1}$$

$$\frac{D}{\sin\alpha} = \frac{x1}{\sin\theta} \rightarrow X1 = \frac{D\ \sin\theta}{\sin\alpha} \tag{2.2}$$

$$X = x1 + x2 = \frac{D\ \sin\gamma}{\sin\beta} + \frac{D\ \sin\theta}{\sin\alpha} \tag{2.3}$$

Em um triângulo a soma de seus ângulos internos é de 180, portanto

$$\alpha + \theta + 90 = 180 \tag{2.4}$$

logo

$$\alpha = 90 - \alpha \tag{2.5}$$

$$\beta + \gamma + 90 = 180 \tag{2.6}$$

$$\gamma = 90 - \beta. \tag{2.7}$$

A partir das equações (3), (4) e (5) obtém-se a expressão,

$$X = \frac{D\ \sin(90-\alpha)}{\sin\alpha} + \frac{D\ \sin(90-\beta)}{\sin\beta} \tag{2.8}$$

e como

$$\sin(90-\alpha) = \cos\alpha \tag{2.9}$$

então

$$X = \frac{D\ \cos\alpha}{\sin\alpha} + \frac{D\ \cos\beta}{\sin\beta}, \tag{2.10}$$

usando a relação

$$\frac{\cos\alpha}{\sin\alpha} = \frac{1}{\tan\alpha} \tag{2.11}$$

e substituindo a relação (11) na equação (10) chega-se a

$$X = \frac{D}{\tan\alpha} + \frac{D}{\tan\beta} = D\,(\frac{1}{\tan\alpha} + \frac{1}{\tan\beta}) \tag{2.12}$$

passando D para o primeiro termo e X para o segundo, temos

$$D = \frac{X}{(\frac{1}{\tan\alpha} + \frac{1}{\tan\beta})}. \tag{2.13}$$

A relação anterior mostra que a distância de um determinado ponto P até a base de um triângulo X pode ser calculada usando os ângulos α e β que formam os segmentos PA e PB com a base desse triângulo. Em um sistema de visão estereoscópica da câmera deve ter a mesma distância focal e estar posicionadas em um vértice da base do triângulo, como mostrado na Figura 5.

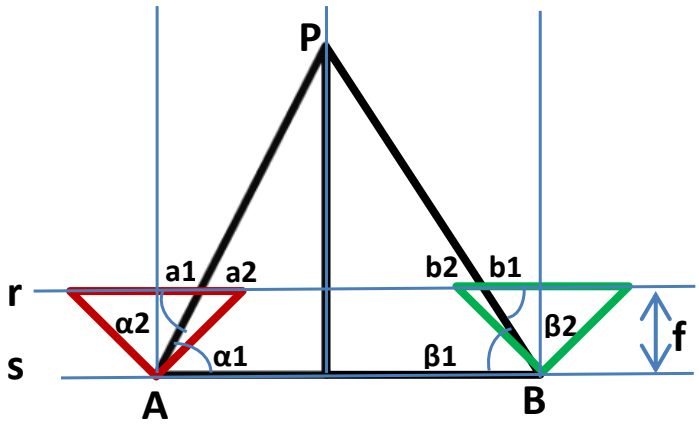


Figura 5 – Simplificação da geometria para formação de imagens estereoscópicas (ESTEVES, 2012).

A equação (13) requere do conhecimento dos ângulos $\alpha 1$ e $\beta 1$, porém tais ângulos não são obtidos de forma direta. Para se adaptar a equação para seu uso direto, algumas adaptações devem ser feitas. A reta r e a reta s são paralelas entre si e, portanto, os ângulos $\alpha 1$ e $\alpha 2$ são iguais da mesma forma que os ângulos $\beta 1$ e $\beta 2$ também são equivalentes. Dessa forma é possível achar as tangentes de α e β pelas seguintes equações,

$$\tan\alpha 2 = \frac{f}{a} \tag{2.14}$$

$$\tan\beta 2 = \frac{f}{b}. \tag{2.15}$$

Utilizando as equações (14) e (15) na equação (13) e considerando $\alpha = \alpha 1 = \alpha 2$ e $\beta = \beta 1 = \beta 2$, obtém-se:

$$D = \frac{X}{(\frac{1}{\frac{f}{a}} + \frac{1}{\frac{f}{b}})} = \frac{X}{\frac{a}{f} + \frac{b}{f}} \tag{2.16}$$

$$D = \frac{X}{\frac{1}{f}(a+b)} = \frac{f\,X}{a+b} \tag{2.17}$$

Os valores de a e b são as distâncias entre o centro da imagem e a localização onde o ponto de interesse está projetado no plano da imagem. Esses valores são absolutos e independem do ponto estar projetado a direita ou a esquerda do centro da imagem.

Assumindo que a largura total de ambos os planos de imagem seja I, o valor de a pode ser substituído por I/2 – a1. Da mesma forma, o valor de b pode ser substituído por I/2 – b1. Usando essas substituições na equação (17), temos:

$$D = \frac{f\,X}{\frac{I}{2} - a1 + \frac{I}{2} - b1} = \frac{f\,X}{I - a1 - b1} \tag{2.18}$$

Para simplificar os cálculos é possível relacionar os valores de a1 e b1 por Xe e Xd respectivamente, onde Xe e Xd representam a distância lateral (medida desde a borda esquerda até o ponto de interesse projetado no plano) das imagens esquerda e direita, respectivamente (vide Figura 6).

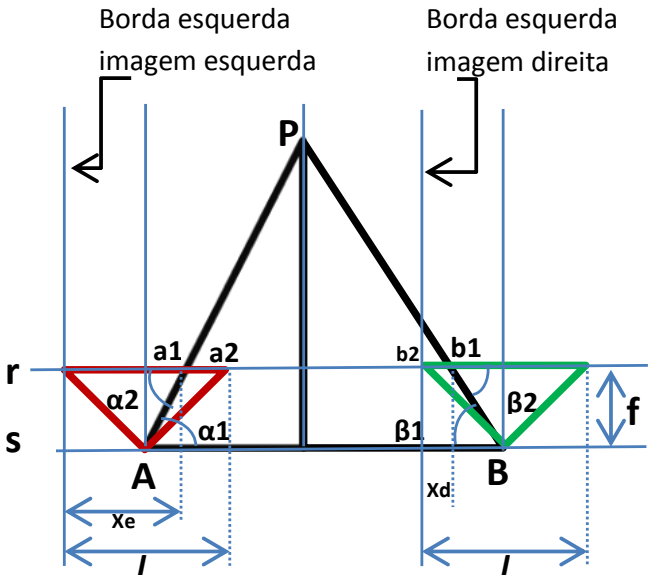


Figura 6 – Projeção do ponto de interesse nas imagens esquerda e direita.

Fazendo uma correspondência entre os modelos das Figuras 5 e 6, temos as seguintes relações:

$$Xe = \frac{I}{2} + \alpha = I - a1 \tag{2.19}$$

$$Xd = \frac{I}{2} - \beta = b1 \tag{2.20}$$

Substituindo os valores de Xe e Xd na equação (18), temos:

$$D = \frac{f\,X}{Xe - Xd} \tag{2.21}$$

O valor Xe–Xd é conhecido como a disparidade entre as imagens. Observe-se que se o ponto P se afasta a disparidade diminui, pois, de acordo com a Figura 4 os ângulos α e β aumentariam e, portanto, os valores de a e de b da Figura 5 diminuem, resultando numa diminuição de Xe e aumento de Xd. Os valores de Xe e Xd são discretos e medidos em pixels, dessa forma conforme um objeto se afasta do par estereoscópico pior se tornará a precisão assim como na visão humana (ESTEVES, 2012).

2.3 Técnicas de Processamento Digital de Imagens Estereoscópicas

As técnicas de processamento digital de imagens estereoscópicas utilizam os mesmos princípios de um processamento digital de imagens padrão, porém, são baseadas no uso de duas imagens no intuito de encontrar correspondências entre elas. Com a posição das partes correspondentes é facilmente calculado a disparidade entre as imagens e assim obter outras informações, tais como, a distância do ponto de interesse ao par estereoscópico.

Para o cálculo de um mapa de disparidade existem basicamente duas aproximações: as baseadas em características e as baseadas em área. Na primeira, os pontos para os quais se determina a disparidade correspondem à posição de determinadas características, como bordas e quinas, nas imagens. Por outro lado, o cálculo da disparidade baseada em área utiliza técnicas de correspondência entre as imagens que usam transformações nas imagens assim como processos de maximização ou minimização de funções custo.

2.3.1 Técnicas de Correspondência por Características

Para o cálculo da disparidade as técnicas de correspondência por características procuram por pontos de interesse nas imagens adquiridas pelo sistema estereoscópico. Esses pontos podem ser bordas, cantos ou arestas que comumente são encontradas usando cálculo de módulos, gradientes e Laplacianas (MARTÍNEZ, 2010). É provável, caso não tenha oclusão entre as imagens, que os pontos característicos de uma imagem existam na outra e, portanto, conhecendo a posição de ambos é possível calcular a disparidade. O cálculo da disparidade de todas as características é conhecido como mapa de disparidade (GONZALES, 2002), (FILHO, 1999). Este mapa de disparidade esta formado pela

diferença das posições de cada característica em uma e outra imagem. O número de correspondências estabelecidas determina a densidade do mapa de disparidade, dependendo, portanto, do número de características encontradas na imagem (COLODRO, 2010).

2.3.2 Técnicas de Correspondência por Área

Diferente das técnicas de correspondência por características que encontram pontos de interesse para o calculo da disparidade, as técnicas de correspondência por área utilizam uma janela que percorre ambas as imagens, e através de funções custo, procura-se pelas janelas mais parecidas entre as imagens (COLODRO, 2010). Com o conhecimento das coordenadas das janelas mais parecidas é possível calcular a disparidade. Das técnicas de correspondência por área mais utilizadas pela comunidade, pode-se mencionar a transformada Rank e a transformada Census.

- **Transformada Rank**

Esta transformada tem como entrada uma matriz que representa uma janela da imagem. A saída representa um valor que corresponde ao número de pixels com intensidade menor que o pixel central. A equação (22) apresenta o modelo matemático da transformada Rank,

$$TR(M) = \sum_{i=0}^{3}\sum_{j=0}^{3} f\left(I_c, I_{i,j}\right) \tag{2.22}$$

onde TR é a transformada Rank, M uma matriz quadrada,$I_{i,j}$ a intensidade do pixel a ser comparado, I_c a intensidade do pixel central e f é uma função de comparação entre os dois pixels c e i,j.

$$f(I_c, I_{i,j}) = \begin{cases} 1, & \text{se } I_{i,j} < I_c \\ 0, & \text{se } I_{i,j} \geq I_c \end{cases}$$

No exemplo mostrado a seguir é calculada a transformada Rank de uma janela 3x3.

$$\mathbf{TR}\left(\begin{bmatrix} 4 & 8 & 15 \\ 16 & 23 & 42 \\ 3 & 7 & 9 \end{bmatrix}\right) = 1+1+1+1+0+1+1+1 = 7$$

As principais desvantagens da transformada Rank são a redução da sensibilidade às distorções radiométricas e a perda de informação devido à falta de uma ordenação dos pixels dentro da janela. Observe-se que a transformada calcula o número de pixels menores entre os pixels da borda da janela ($I_{i,j}$) e o pixel central (I_c). Portanto, após a

transformação, os pixels de uma janela são codificados com um único valor. No intuito de aumentar a capacidade de discriminação de correspondência entre as imagens, Zabihe Woodfill realizaram uma modificação baseada na preservação da ordem dos pixels. Essa nova transformada foi denominada de Transformada Census (MARTÍNEZ, 2010).

- **Transformada Census**

Do mesmo modo que a transformada Rank, esta transformada Census compara a intensidade dos pixels de uma janela qualquer com o pixel central desta janela, porém,é realizada uma concatenação que visa manter a ordem dos pixels maiores e menores. Esta comparação está representada pela seguinte função.

$$TC(M) = F(I_c, I_{i,j}) \tag{2.23}$$

$$F(I_c, I_{i,j}) = \begin{cases} 0, & \text{se } I_{i,j} > I_c \\ 1, & \text{se } I_{i,j} \leq I_c \end{cases}$$

onde $I_(i,j) \in D_(i,j)$ e o pixel i,j é diferente pixel central, isto é, não se realiza a comparação do pixel central com ele mesmo. TC é a transformada Census, M é uma matriz quadrada, $I_(i,j)$ é a intensidade do pixel em comparação, I_c é a intensidade do pixel central e F é uma função de comparação entre os dois pixels c e os pixels vizinhos i, j. No exemplo mostrado a seguir é calculada a transformada Census de uma janela 3x3

$$\vec{\mathbf{TC}}\left(\begin{bmatrix} 4 & 8 & 15 \\ 16 & 23 & 42 \\ 3 & 7 & 9 \end{bmatrix}\right) = 11110111$$

2.3.3 Funções Custo usadas nas técnicas de Correspondência por Área

Uma forma de realizar a correspondência entre duas imagens estereoscópicas é mediante o uso direto de funções custo que minimizem a diferença de intensidade entre os pixels de uma imagem com relação aos pixels da outra imagem. As funções custo mais comumente usadas para o cálculo de correspondência são descrita a seguir.

- **SAD**

A função custo SAD (soma das diferenças absolutas) possui uma janela como entrada e entrega um valor inteiro como saída. Segundo França (FRANÇA, 2003) esta função é muito comum em casos onde o tempo de processamento é crítico, e pode ser expressa pela equação abaixo.

$$SAD(M_1, M_2) = \sum_i \sum_j |I_{1(x+i,y+j)} - I_{2(x+i+d,y+1)}| \tag{2.24}$$

onde M é uma matriz quadrada, d é o valor da disparidade do ponto analisado, i e j são indicadores de varredura das janelas,$I_{2(x,y)}$é a intensidade de um ponto da imagem da janela de busca,$I_{1(x,y)}$é a intensidade de um ponto da imagem da janela alvo. No exemplo a seguir é mostrado o cálculo da função SAD para duas janelas 3x3.

$$\textbf{SAD}\left(\begin{bmatrix} 2 & 1 & 3 \\ 6 & 5 & 2 \\ 3 & 1 & 4 \end{bmatrix}, \begin{bmatrix} 5 & 1 & 2 \\ 2 & 1 & 2 \\ 3 & 1 & 3 \end{bmatrix}\right) = |-3| + 0 + 1 + 4 + 4 + 0 + 0 + 0 + 1 = 13.$$

- **SSD**

A função custo SSD (soma dos quadrados das diferenças) é representada pela equação (25).

$$SSD(M_1, M_2) = \sum_i \sum_j (I_{1(x+i,y+j)} - I_{2(x+i+d,y+1)^2} \tag{2.25}$$

onde M é uma matriz quadrada, d é o valor da disparidade do ponto analisado, i e j são indicadores de varredura das janelas $I_{2(x,y)}$ é a intensidade de um ponto da imagem da janela busca e $I_{1(x,y)}$ é a intensidade de um ponto da imagem da janela alvo. No exemplo a seguir é mostrado o cálculo da função SSD para duas janelas 3x3.

$$\textbf{SAD}\left(\begin{bmatrix} 2 & 1 & 3 \\ 6 & 5 & 2 \\ 3 & 1 & 4 \end{bmatrix}, \begin{bmatrix} 5 & 1 & 2 \\ 2 & 1 & 2 \\ 3 & 1 & 3 \end{bmatrix}\right) = 9 + 0 + 1 + 16 + 16 + 0 + 0 + 0 + 1 = 43.$$

- **Distancia de Hamming**

A função custo distância de Hamming, comumente utilizada após a transformação Census, tem como objetivo representar a correspondência entre as posições de duas imagens. A distância de Hamming realiza uma comparação bit a bit entre as duas janelas da imagem de referência e da imagem alvo. O resultado é um valor numérico que representa quantos destes bits são diferentes.

$$H(\vec{V_1}, \vec{V_2}) = \sum_j f(V_1, V_2) \tag{2.26}$$

$$f(\vec{V_1}, \vec{V_2}) = \begin{cases} 0, & \text{se } \vec{V_1} > \vec{V_2} \\ 1, & \text{se } \vec{V_1} \neq \vec{V_2} \end{cases}$$

No exemplo a seguir é mostrado o cálculo da distância de Hamming para duas janelas 3x3.

$$H(00110011, 11110000) = 1 + 1 + 0 + 0 + 0 + 0 + 1 + 1 = 4$$

2.4 Hardware Reconfigurável

Desde a introdução do primeiro microprocessador comercial, o Intel 4004, no final do ano 1971, os sistemas digitais têm evoluído de forma considerável. Este pequeno microprocessador integrava 2300 transistores em uma única pastilha de silício cujo custo inicial oscilava os 200 dólares americanos. A complexidade dos microprocessadores, medida segundo o número de transistores dentro do chip, é dobrada cada 18 meses desde a aparição do 4004 (Moore, 1997). Esta observação, conhecida como lei de Moore, tem se mostrado válida para qualquer gama de circuitos digitais. Com a evolução da tecnologia os circuitos integrados conseguem atualmente integrar cada vez mais transistores, atingindo alguns bilhões nos dias atuais.

Uma classificação dos circuitos integrados que permitem a implementação de uma lógica digital é mostrada na seguinte figura.

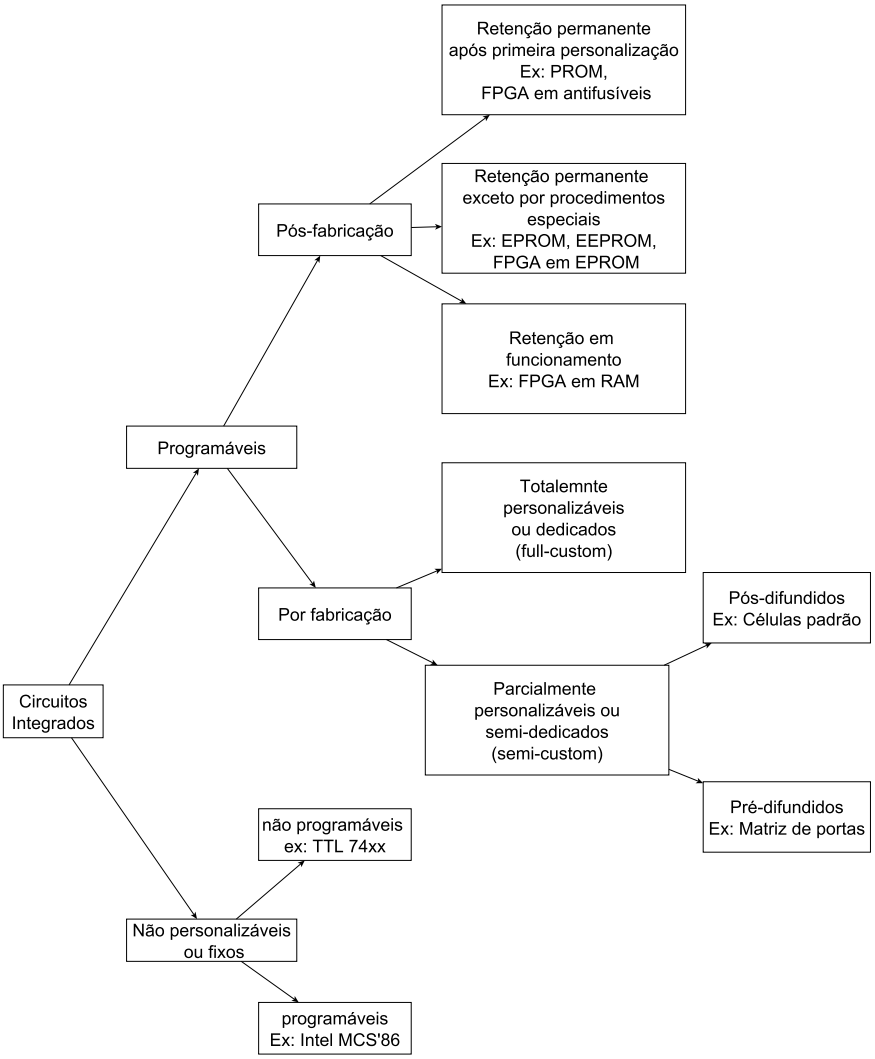


Figura 7 – Taxonomia para CIs (modificado (CALAZANS, 1998)).

Os circuitos integrados não personalizáveis executam uma lógica padrão realizando operações pré-definidas pelo fabricante. A tecnologia VLSI (*Very Large Scale Integration*), definida como uma tecnologia de integração de alta escala, permite ao usuário determinar a lógica a ser implementada em um circuito. Entre os circuitos integrados projetados usando a tecnologia VLSI destacam-se os ASICs (*Application Specific Integrated Circuits*) nos quais os circuitos integrados são personalizávies, porém, construídos para tarefas específicas. Entre as metodologias de projeto de ASICs podem-se mencionar o projeto por células padrão (standard cells), matriz de portas (gate arrays), projeto estruturado (structured array) e projeto totalmente customizado (full-custom design) (SMITH, 1997), (WU, 2004). Por outro lado, o tempo de projeto e fabricação de ASICs eleva o custo de desenvolvimento, justificado o investimento para uma grande quantidade de circuitos integrados. Contudo, este tipo de tecnologia alcança velocidades de processamento altas em uma área reduzida, assim como um baixo consumo de energia.

PLDs (*Programmable Logic Devices*) são circuitos construídos a partir de um ar-

ranjo matricial de elementos lógicos reprogramáveis, permitindo implementar, teoricamente, qualquer tipo de circuito digital. Os PLDs não possuem uma estrutura fixa eas sua funções são definidas pelo usuário. Alguns exemplos de PLDs são PROM (*Programmable Read Only Memory*), PLA(*Programmable Logic Array*), FPGA e CPLD (*Complex Programmable Logic Device*) (MUNOZ, 2012).

Para se descrever o comportamento de um circuito eletrônico usando as tecnologias de matriz de portas e dispositivos PLDs deve-se utilizar uma linguagem de descrição de *hardware* como o VHDL ou Verilog. A linguagem VHDL foi desenvolvida para substituir os complexos manuais que descreviam o funcionamento dos ASICs, e posta em domínio público e padronizada pela IEEE no ano de 1987 (PEDRONI, 2004).

2.4.1 FPGAs

Dispositivos FPGAs são formados por blocos lógicos configuráveis conectados através das chaves de interconexão, e podem ser configuradas pelo usuário para implementar aplicações específicas. Os blocos lógicos configuráveis podem conter vários elementos lógicos, e cada elemento lógico é formado por um número determinado de *Look Up Tables* (LUTs), multiplexadores e registradores. Os LUTs são formados por 4, 5, 6 ou 8 entradas, dependendo da arquitetura e permitem implementar a tabela verdade de uma função lógica booleana. Adicionalmente, os FPGAs possuem outros componentes como memórias RAM (*Random Access Memory*), PLL (*Phase Locked Loop*), DSPs(*Digital Signal Processing*), DCMs (*Data Clock Managment*), entre outros, que permitem maior flexibilidade das implementações realizadas em *hardware*.

É importante ressaltar que todos os elementos internos do FPGA podem ser acessados de forma paralela no intuito de acelerar a execução dos algoritmos implementados. A Figura 8 mostra a arquitetura interna de um FPGA da Xilinx.

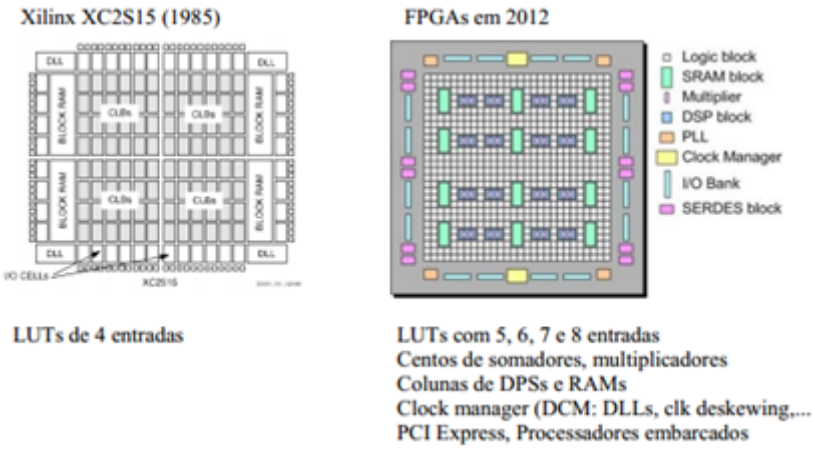


Figura 8 – Estrutura interna de um FPGA.

2.4.2 Fluxo de Projeto em FPGAs

A Figura 9 mostra o fluxo geral de projeto em FPGAs. O fluxo está baseado em sete etapas descritas a seguir.

- Especificação funcional: Nesta etapa faz-se a análise de requisitos do projeto de *hardware*.
- Descrição do *hardware*: Esta etapa consiste na implementação do através de HDLs (*Hardware Description Language*).
- Etapa de simulação comportamental: processo de otimização da lógica implementada pelo HDL. Durante a etapa de simulação comportamental avalia-se o comportamento da descrição de *hardware* realizada anteriormente, caso esta análise tenha resultados satisfatórios passa-se para a próxima etapa, caso contrario reformula-se a descrição de *hardware*.
- Síntese lógica: utiliza algoritmos de síntese lógica para representar a descrição de *hardware* usando funções booleanas.
- Place and route: realiza o mapeamento das funções booleanas nas LUTs e DSPs e memorias RAM, entre outros, assim como a interconexão entre os diferentes elementos.
- Etapa de simulação funcional ou elétrica: processo de verificação da lógica e dos requisitos de *timming* e consumo de potência da arquitetura de *hardware*. Caso a verificação tenha resultados satisfatórios passa-se para a próxima etapa.
- Prototipagem em dispositivos FPGAs: Após todo processo anterior realiza-se a prototipagem do projeto e tem-se todos os resultados reais, como por exemplo, consumo, quantidade de DSP e LUTs utilizados, frequência de operação, entre outros.

2.5 Publicações sobre a transformada Census

Para fins de referência fez-se um levantamento de publicações sobre a transformada Census e suas modificações.

Qiu realizou três transformadas Census, sendo uma para cada canal de cor (RGB) obtendo, assim, resultados mais precisos na extração de bordas de objetos móveis em uma imagem (QIU, 1998).

Gautama faz um estudo sobre a transformada Census, transformada Census modificada e a correlação por media normalizada para realizar a observação da cabine de um carro para obter informações sobre os passageiros e assim ativer ou não o airbag. A

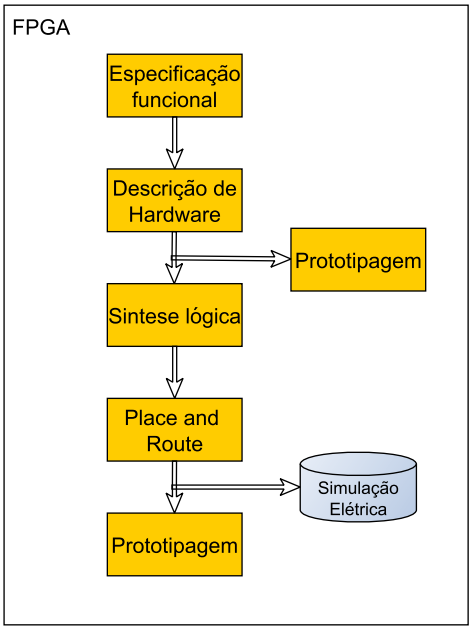


Figura 9 – Fluxo de projeto em FPGAs

modificação usada na transformada Census leva em consideração o valor médio da janela para as comparações (GAUTAMA, 1999).

Woodfill implementou o algoritmo da transformada Census para o calculo de disparidade em uma FPGA e conseguiu assim uma velocidade superiores a 200 frames por segundo em uma imagem 320x240 (WOODFILL, 1999).

Yamada utiliza a transformada Census para o cálculo da disparidade através de três câmeras (YAMADA, 2000).

(KUHN, 2003) combina a transformada Census e a Soma dos Quadrados das Diferenças e análisa os resultados.

Woodfill implementou a transformada Census em uma ASIC e utilizou um processador (Deep Sea Board) para comunicar com o ASIC, obtendo, assim, 200 frames por segundo em uma imagem de 512x480 (WOODFILL, 2004).

Além das modificações da transformada Census, fez-se um levantamento sobre a utilização da FPGA com a transformada Census, entre eles, (NAOULOU, 2006) e em 2007 Murphy et al implementou um sistema de visão estéreo de baixo custo implementado em FPGA, que utiliza a Transformada Census em seu algoritmo de correlação.

Ibarra implementou a transformada Census e SAD em uma FPGA e analisou/comparou os resultados performance obtidos (IBARRA, 2009).

Colodro fez uma comparação entre algoritmos de correspondência, entre eles, Rank 7x7, Mediana 5x5, Census 9x9, CensusE 5x5, SAD 11x11 (COLODRO, 2010).

Hamza implementou um algoritmo de detecção de obstáculos em uma FPGA e comparou com microprocessadores e com *software*, obtendo resultados até cem vezes melhores (HAMZA, 2012).

3 Implementação

Para a implementação das arquiteturas foi adquirido um kit Digilent Atlys que possui um dispositivo FPGA Xilinx Spartan-6 cuja tecnologia utiliza slices (blocos lógicos configuráveis) com 4 LUTs de 6 entradas e 8 registradores cada. Adicionalmente, o kit Atlys possui um conector VHDCI (*very-high-density cable interconnect*) que permite a comunicação entre o FPGA e periféricos externos. Por outro lado, também foi adquirido um kit de visão estéreo, chamado de VmodCAM. O VmodCAM possuí câmeras com 63mm de espaçamento entre as câmeras e distância focal de 3.81mm, além de permitir uma resolução de 1600x1200 a 15fps (*frames* por segundo).

É importante salientar que na arquitetura desenvolvida foram usadas duas memórias ROM, uma para cada imagem, visando emular o fluxo de pixels das câmeras esquerda e direita. Isto possibilita a verificação modular do circuito sem o uso direto do sistema estereoscópico (VmodCAM) e sua correspondente calibração.

3.1 Arquitetura Geral

A Figura 9 representa a arquitetura geral para a solução do problema proposto. Nela estão representadas todas as etapas para a obtenção de um mapa de disparidade. As duas imagens, esquerda e direita, estão representadas cada uma por uma memória ROM. Cada imagem é então passada para próxima etapa denominada de máscara *buffer* na qual são usados registradores dedicados para armazenar uma porção da imagem permitindo o posterior deslocamento da máscara a cada ciclo de relógio. Posteriormente, é realizado a transformada Census das imagens esquerda e direita. Em seguida a imagem Census direita é armazenada em registradores dedicados (Buffer da Transformada Census) facilitando a comparação entre os pixels da mesma linha epipolar. Após esta etapa é feito o cálculo, em uma abordagem paralela, de todas as distâncias de Hamming entre os pixels da mesma linha epipolar da imagem esquerda e direita. Finalmente, é calculado, usando uma arquitetura *pipeline*, o valor mínimo entre todas as distâncias de Hamming obtidas do passo anterior e como resultado tem-se o mapa de disparidade desta linha epipolar.

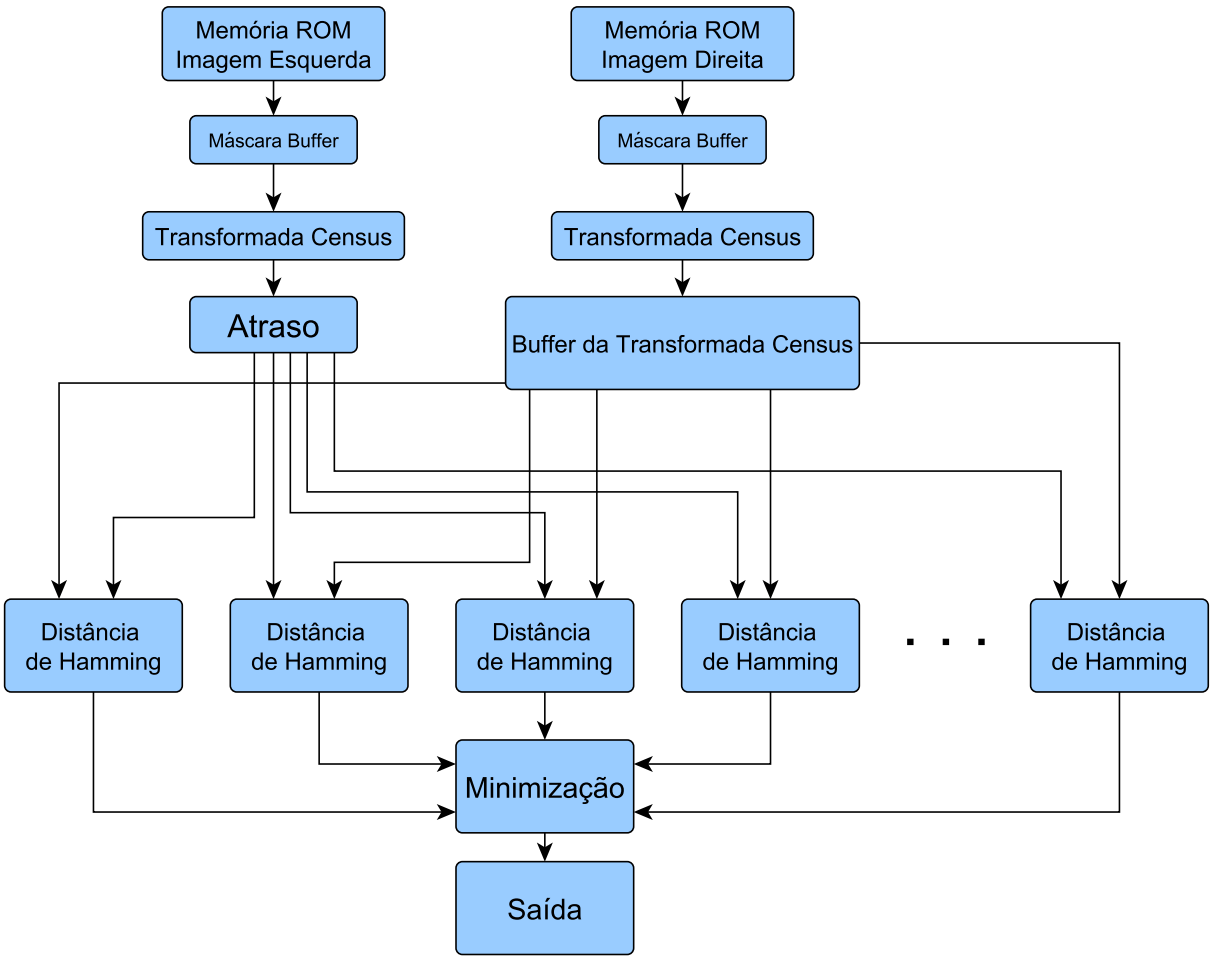


Figura 10 – Arquitetura pipeline Geral da implementação em *hardware*

3.2 Máscara Buffer

No processamento de imagens no domínio espacial, comumente se trabalha com janelas de tamanho pré-definido sobre as quais são realizadas operações de vizinhança. A arquitetura *Máscara Buffer* é responsável pelo armazenamento e deslocamento dos pixels de entrada de forma que uma vizinhança (ou janela) possa ser disponibilizada em um único ciclo de clock.

A figura 11 apresenta a arquitetura proposta para o processo de carregamento da vizinhança. Em particular, apresenta-se a arquitetura para o carregamento de uma janela de tamanho 3x3. O pixel proveniente da câmera entra no canto esquerdo superior, sendo armazenado no primeiro registrador. Quando o segundo pixel é enviado pela câmera então o primeiro pixel é deslocado à direita. O processo continua até completar a primeira linha. Com a chegada de um novo pixel, o primeiro pixel é deslocado para a primeira posição da segunda linha, continuando com o processo de armazenamento e deslocamento de pixels. Uma vez completado o envio de duas linhas e três pixels, será possível disponibilizar a cada

ciclo de clock uma nova janela de 3x3. Observe-se que para janelas de 5x5 seria necessário armazenar quatro linhas e cinco pixels e, de forma geral, quanto maior o tamanho da janela de operação, maior será o número de registradores necessários.

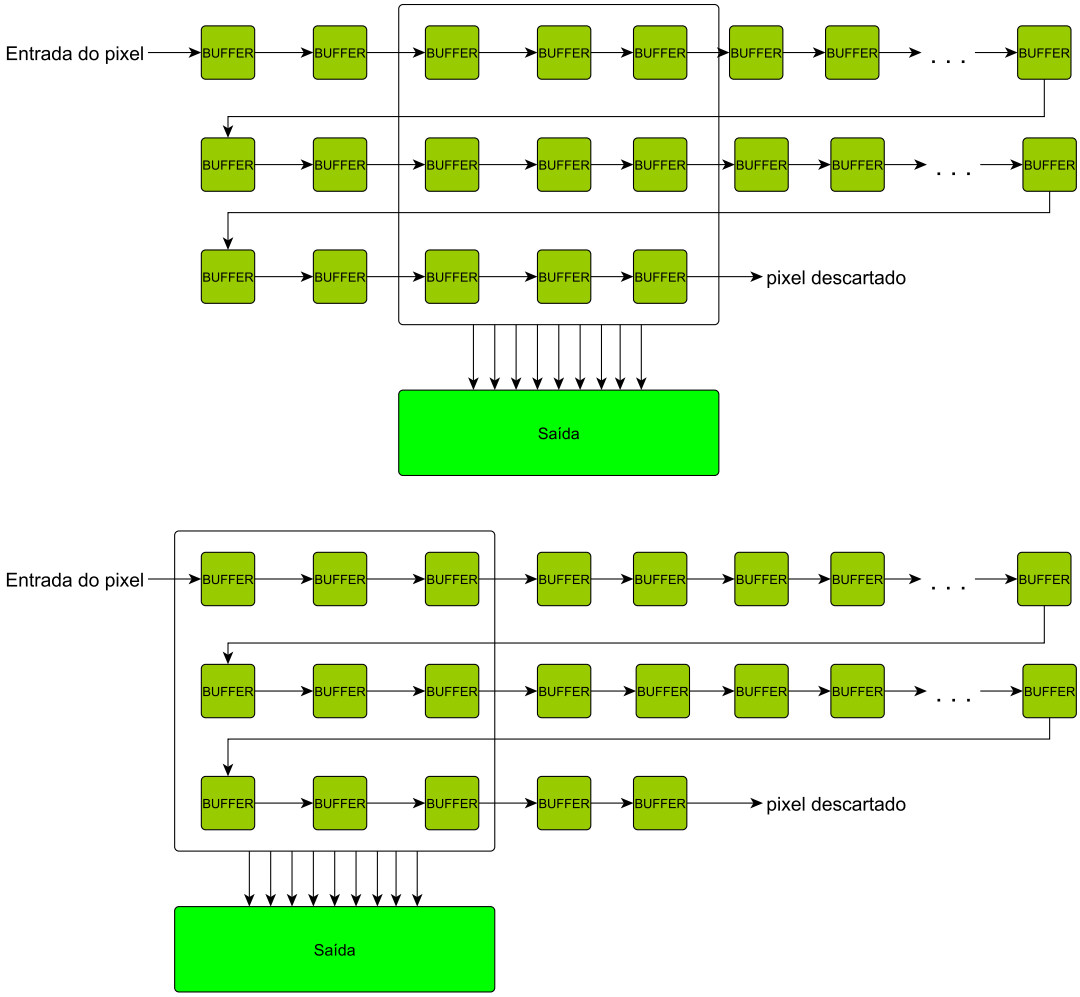


Figura 11 – Arquitetura da Máscara *Buffer* para uma janela 3x3.

Para mudar de linha a máscara demora três pulsos de relógio e deveria ser feito em apenas um, portanto, para corrigir este problema em todas troca de linha a máscara move-se de posição da direita para esquerda.

3.3 Transformada Census

A figura 12 apresenta a arquitetura de *hardware* proposta para o cálculo da transformada Census. Na figura é apresentado o caso particular de uma janela 3x3 disponibilizada pelo módulo Máscara Buffer. Esta arquitetura está baseada em comparações paralelas entre o pixel central (I5) e seus pixels vizinhos, sendo o resultado destas comparações disponibilizado em forma de um vetor de saída, chamado de *pixel Census*. Assim, o

deslocamento da janela através das imagens esquerda e direita resulta nas imagens Census esquerda e direita, respectivamente.

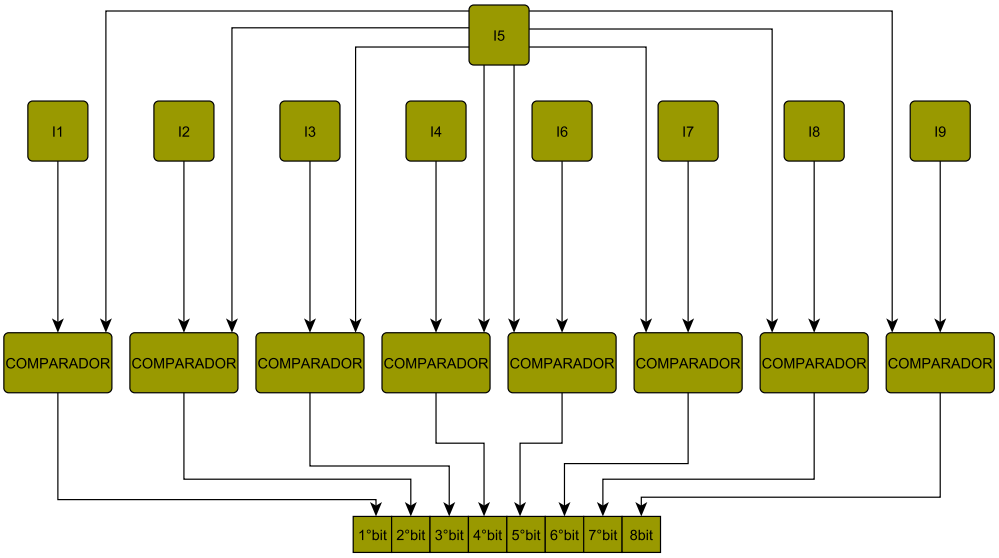


Figura 12 – Arquitetura de *hardware* proposta para a transformada Census.

3.4 Buffer da Transformada Census

No cálculo de correspondências entre duas imagens Census, o objetivo é encontrar um pixel particular de uma imagem na outra imagem. A diferença entre a posição destes pixels estará relacionado com a disparidade entre as imagens. Assim, considerando uma linha epipolar, um ponto na imagem esquerda pode estar na mesma coordenada da imagem direita ou em uma coordenada à sua esquerda. A arquitetura *buffer da transformada Census* realiza o armazenamento e deslocamento dos píxels da imagem Census direita. Desta forma é possível comparar um *píxel census* localizado em alguma coordenada da imagem esquerda com todos os *píxel census* à esquerda do pixel localizado da mesma coordenada da mesma linha epipolar na imagem direita.

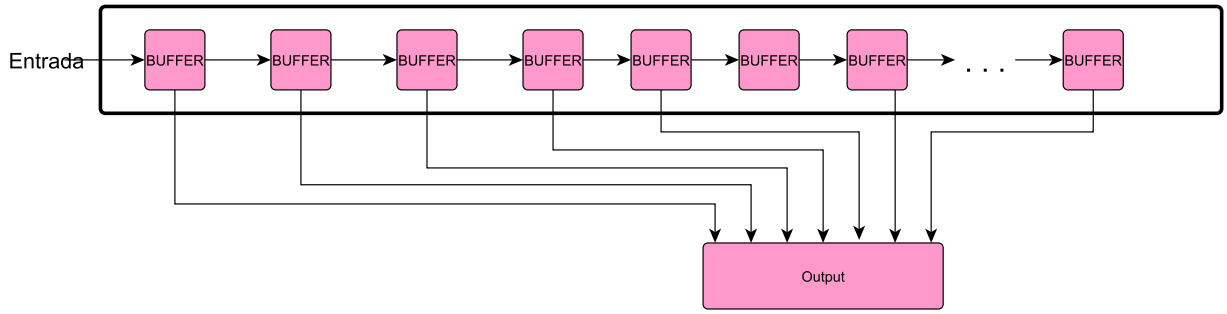


Figura 13 – Buffer da transformada Census.

3.5 Distância de Hamming

A distância de Hamming é baseada em comparações bit a bit de cada vetor resultante da transformada Census, na qual bits com valores diferentes resultam em 1 e bits iguais em 0, seguida de um somatório dessas comparações. A figura 13 apresenta a arquitetura proposta para o cálculo da distância de Hamming entre dois *píxel census*. A arquitetura realiza em forma paralela a comparação dos bits de cada pixel. Conforme mostrado na Figura 10, esta arquitetura é instanciada diversas vezes em paralelo, visando calcular a distância de Hamming entre cada *píxel census* da imagem esquerda em uma coordenada com os *píxel census* à esquerda da mesma coordenada da imagem direita.

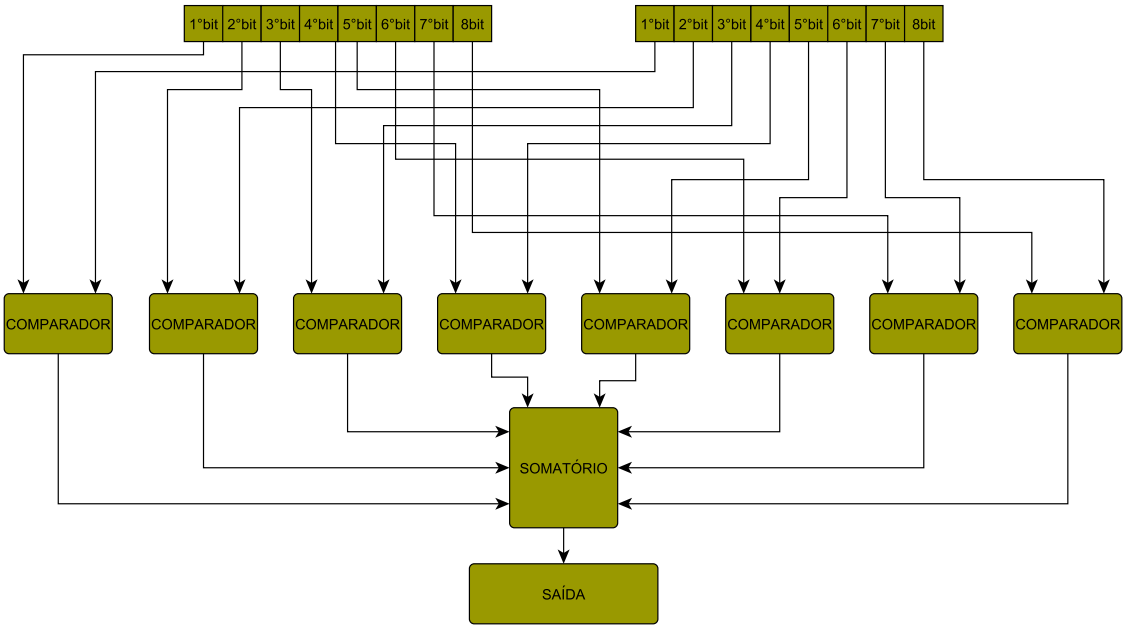


Figura 14 – Arquitetura da distância de Hamming.

3.6 Minimização

O cálculo da distância de Hamming é feito comparando um pixel da imagem esquerda como primeiro pixel da imagem direita até o pixel com a mesma coordenada do pixel esquerdo, após o cálculo de todas estas distâncias de Hamming é necessário obter o menor valor entre eles, portanto, é feito uma minimização, ou seja a comparação de todos os valores escolhendo o menor entre eles. Para tanto o *hardware* proposto na figura 15 faz comparações de dois a dois e em cadeia. A latência gerada por esta arquitetura é igual a $\log_2(Ncolunas + 1 - Tmascara)$, onde Ncolunas é o números de colunas da imagem e Tmascara é a máscara utilizada.

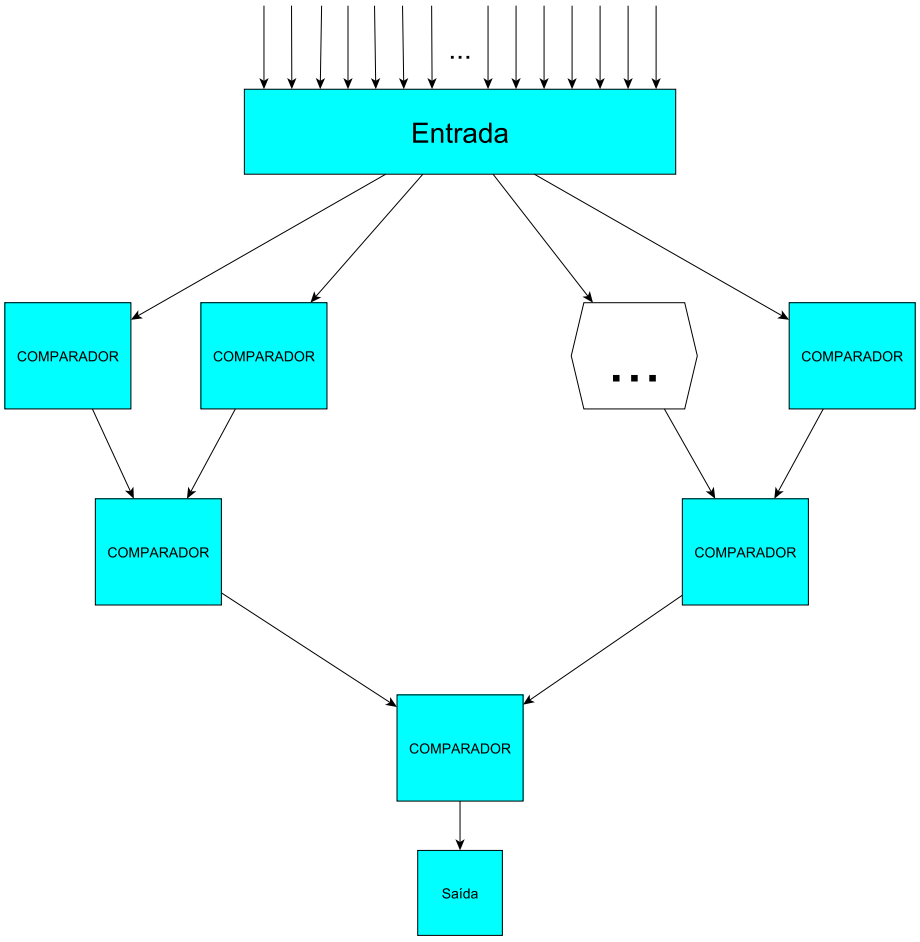


Figura 15 – Arquitetura da minimização.

4 Resultados

Este capítulo apresenta os resultados obtidos na simulação em *software* e na implementação em textithardware. Para a simulação em *software* é apresentado uma análise de *profile* em relação ao tempo de execução para se justificar a implementação em *hardware*. Os resultados na implementação de *hardware* são mostrados através de figuras comparativas e por tabelas de síntese.

4.1 Resultados de Síntese

Os resultados de síntese lógica das arquiteturas desenvolvidas para as máscaras 3x3, 5x5, 7x7 e 9x9 estão representadas nas tabelas abaixo. Estes resultados permitem analisar o consumo de recursos de *hardware* dos circuitos desenvolvidos, assim como a frequência de operação dos mesmos.

3x3	Resultados			
	LUTs(27288)	FFs(54576)	Freq(MHz)	RAM
Completa	4521(16%)	6435(11%)	197.6	4
Máscara Buffer	1324	1987	200	–
Transformada Census	41	–	–	–
Buffer da TC	338	736	419.2	–
Distância de Hamming	16	22	887	–
Minimização	1352	596	314.5	–

Tabela 1 – Resultados de síntese para janela 3x3

5x5	Resultados			
	LUTs(27288)	FFs(54576)	Freq(MHz)	RAM
Completa	11253(41%)	12681(23%)	193.5	4
Máscara Buffer	3246	3818	192	–
Transformada Census	121	–	–	–
Buffer da TC	324	1539	419,2	–
Distância de Hamming	70	67	572	–
Minimização	745	567	341	–

Tabela 2 – Resultados de síntese para janela 5x5

7x7	Resultados			
	LUTs(27288)	FFs(54576)	Freq(MHz)	RAM
Completa	21961(80%)	20993(38%)	210.1	4
Máscara Buffer	6994	6172	208	–
Transformada Census	241	–	–	–
Buffer da TC	310	2679	419.2	–
Distância de Hamming	121	110	536	–
Minimização	814	594	262	–

Tabela 3 – Resultados de síntese para janela 7x7

9x9	Resultados			
	LUTs(27288)	FFs(54576)	Freq(MHz)	RAM
Completa	33828(123%)	30775(56%)	204.2	4
Máscara Buffer	10728	8193	205	–
Transformada Census	474	–	–	–
Buffer da TC	296	4107	419.2	–
Distância de Hamming	197	180	531	–
Minimização	908	617	282	–

Tabela 4 – Resultados de síntese para janela 9x9

11x11	Resultados			
	LUTs(27288)	FFs(54576)	Freq(MHz)	RAM
Completa	42253(154%)	37442(68%)	188	4
Máscara Buffer	12926	10277	201	–
Transformada Census	625	–	–	–
Buffer da TC	282	5774	419.2	–
Distância de Hamming	292	253	531	–
Minimização	869	604	282	–

Tabela 5 – Resultados de síntese para janela 11x11

Pode-se observar nas tabelas 1, 2, 3, e 4 que o aumento da máscara utilizada ocasionou em um crescimento quase exponencial no consumo de LUTs e FFs, porém a frequência de operação manteve-se acima da faixa estipulada de 180MHz. Para a máscara de 9x9 observa-se que os recursos disponíveis de LUTs foram extrapolados, isto ocorre devido a arquitetura de deslocamento de pixel (máscara buffer), em que devido a imagem final possuir tamanho menor que a imagem original foi feito uma lógica de armazenamento para realizar esta lógica e permanecer com um resultado a cada pulso de *clock*, porém a melhor solução seria preencher a imagem com uma borda preta, obtendo uma imagem com o mesmo tamanho da original.

4.2 Validação das Arquiteturas

Neste tópico é mostrado os resultados da implementação de *hardware* através das comparações entre as imagens do banco de dados da Middlebury com as imagens obtidas e através de uma comparação númerica utilizando o erro médio quadrático, além disso, é feito uma comparação das imagens obtidas em *software* e *hardware*.

4.2.1 Resultados de simulação

Para as simulações em *hardware* foram utilizadas as imagens *sawtooth* do banco de dados da Middlebury (MIDDLEBURY, 2001). Para imagem esquerda utilizou-se como base a imagem im2.ppm, figura 16, e para a imagem direita foi utilizada imagem im6.ppm, figura 17, entretanto, para as simulações, foi necessário reduzir as imagens para um tamanho de 56x56 pixels para se adequar às limitações do projeto, especificamente com relação às capacidades de memória embarcada no chip FPGA escolhido.

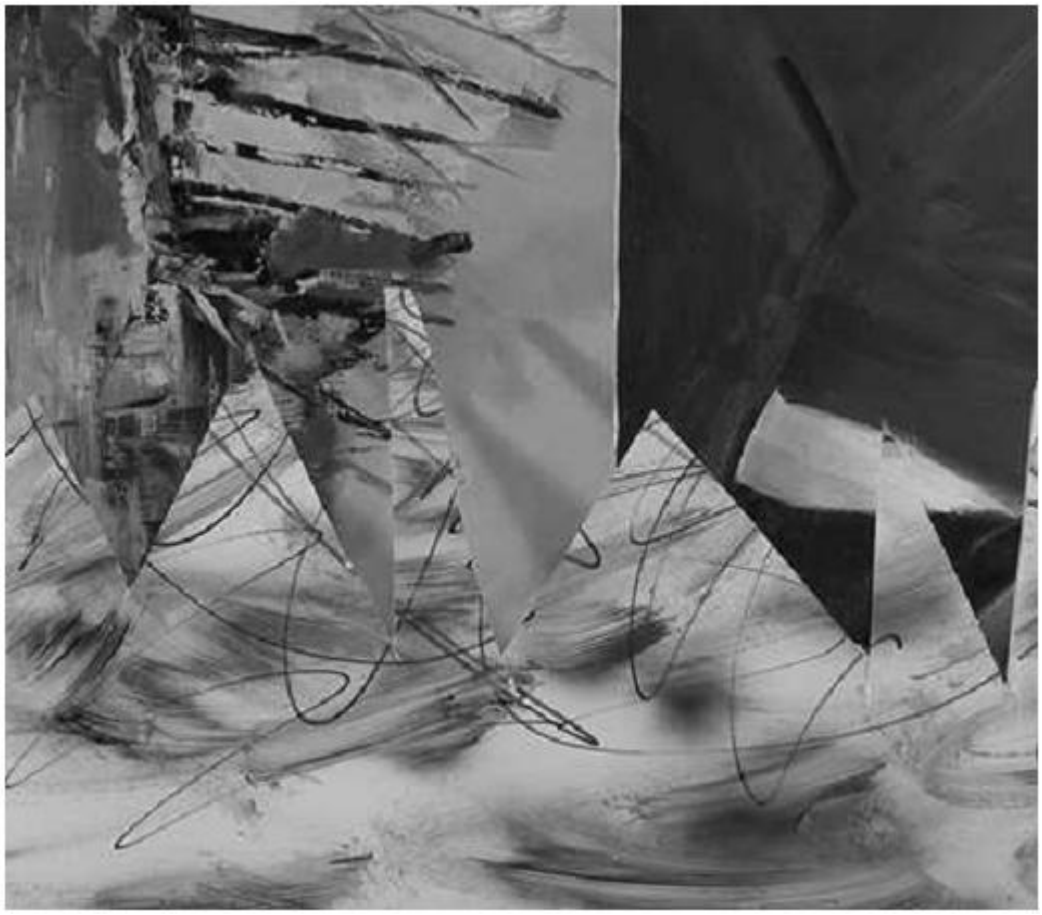


Figura 16 – Imagem im2.ppm obtida do banco de dados da Middlebury

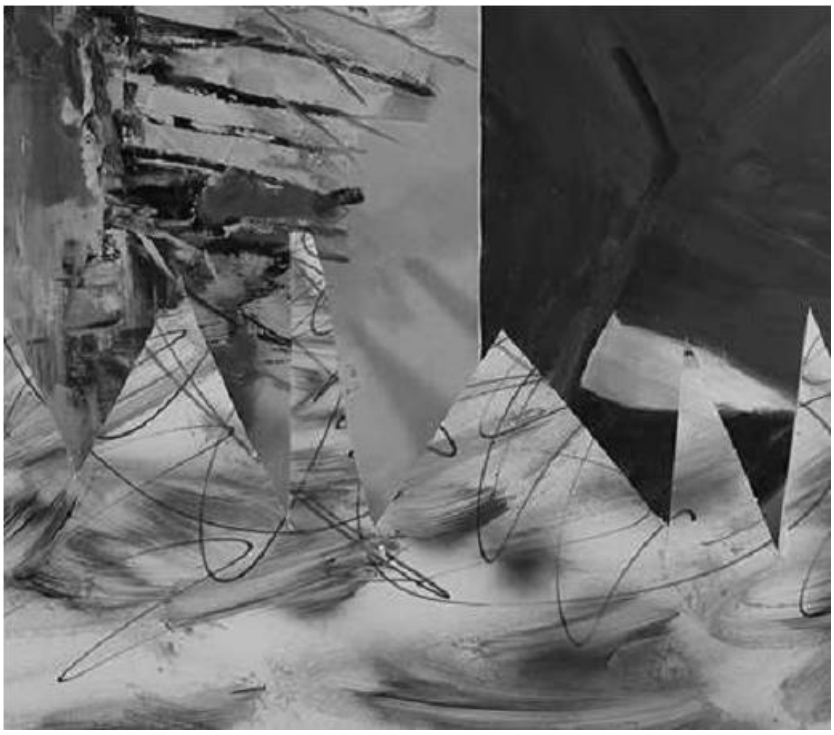


Figura 17 – Imagem im6.pmm obtida do banco de dados da Middlebury

As imagens de tamanho 56x56 cortadas das imagens acima estão representadas pela figura 18, sendo (A) a imagem esqueda e (B) a imagem direita.

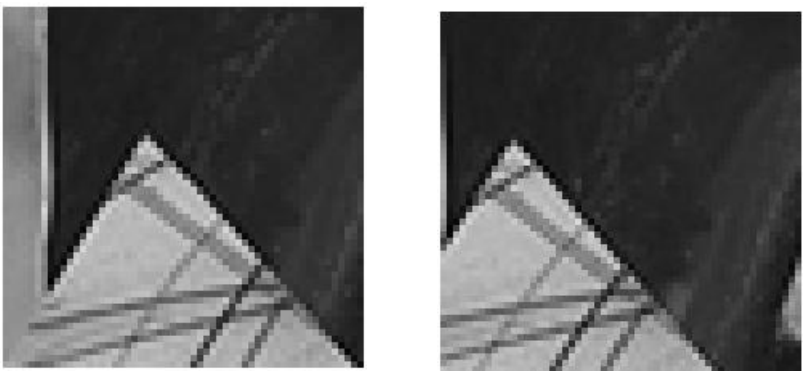


Figura 18 – Cortes de tamanho 56x56 da imagem original: (A)Imagem esquerda e (B)Imagem direita

Através destas duas imagens 56x56, esquerda e direita, foi possível obter os mapas de disparidade para as máscaras. A Figura 19 mostra os mapas de disparidades obtidos usando as máscaras 3x3, 5x5, 7x7, 9x9 e 11x11. Para efeitos de visualização todos os mapas de disparidades obtidos foram multiplicados por dez, portanto, a cada valor 10 no mapa de disparidade representa uma disparidade de 1.

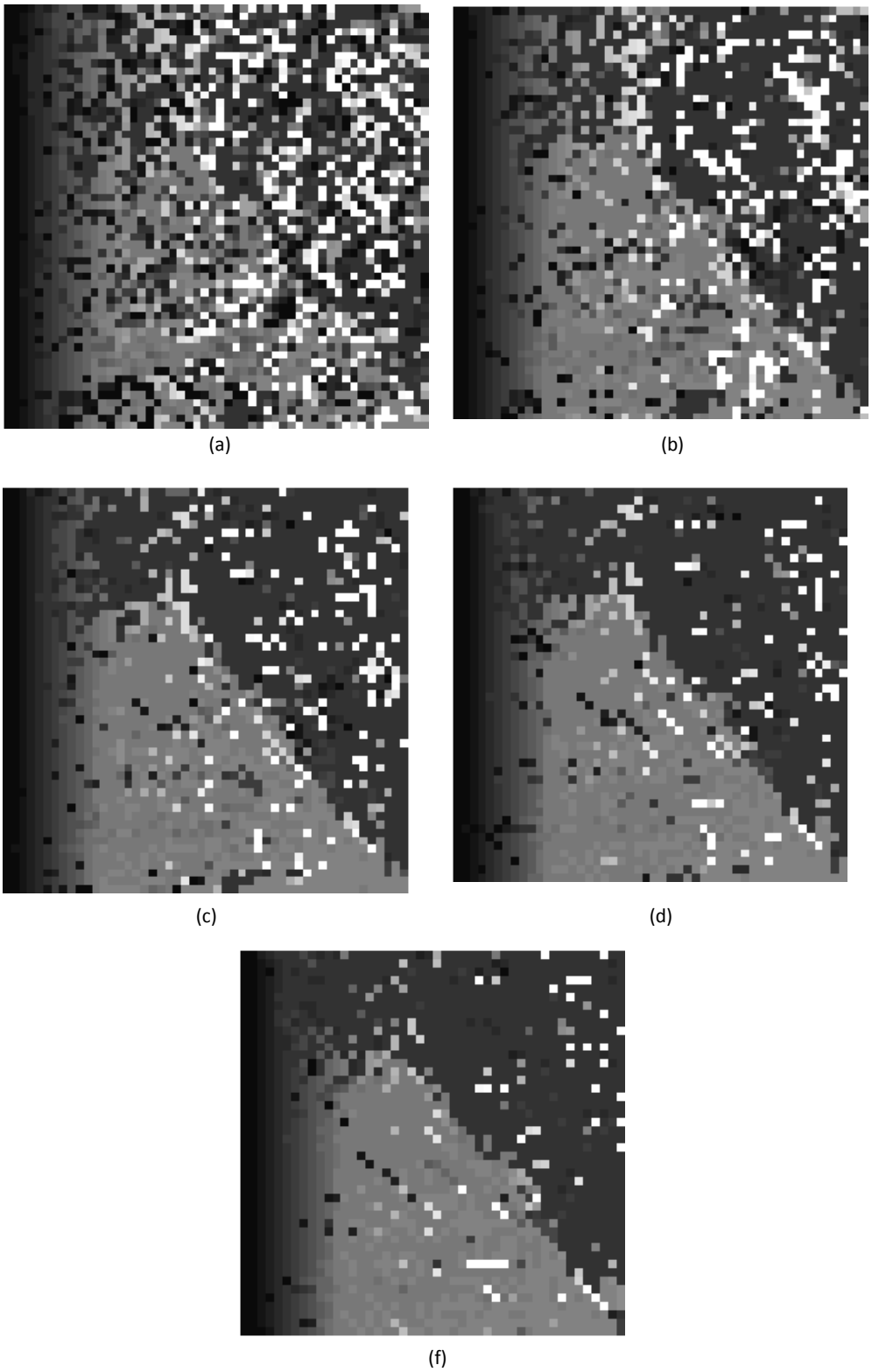


Figura 19 – Mapa de disparidade da arquitetura em *hardware* para as máscaras (a) 3x3, (b) 5x5, (c) 7x7, (d) 9x9, (e) 11x11

4.2.2 Comparação HW/SW

Para uma implementação em *hardware* de uma máscara 3x3 sobre uma imagem 56x56 obteve-se uma latência inicial de 232 pulsos de relógio, após esta latência inicial, um píxel de saída será processado por cada ciclo de relógio, ou seja, mais 2916 pulsos de relógio para finalizar a imagem, totalizando uma latência total de 3148 pulsos de relógio. Sabe-se através da tabela 1 que a frequência de operação para esta arquitetura é de 197Mhz, portanto, o tempo total de execução é de 15,98us. Para uma implementação em *software* da mesma máscara e imagem obteve-se um tempo de execução de aproximadamente 3,37ms, portanto a arquitetura de *hardware* teve um fator de aceleração de 211 vezes. Os mapas de disparidade para a arquitetura de *hardware* e *software* citadas podem ser vistas na figura 20 (a) e (b) respectivamente.

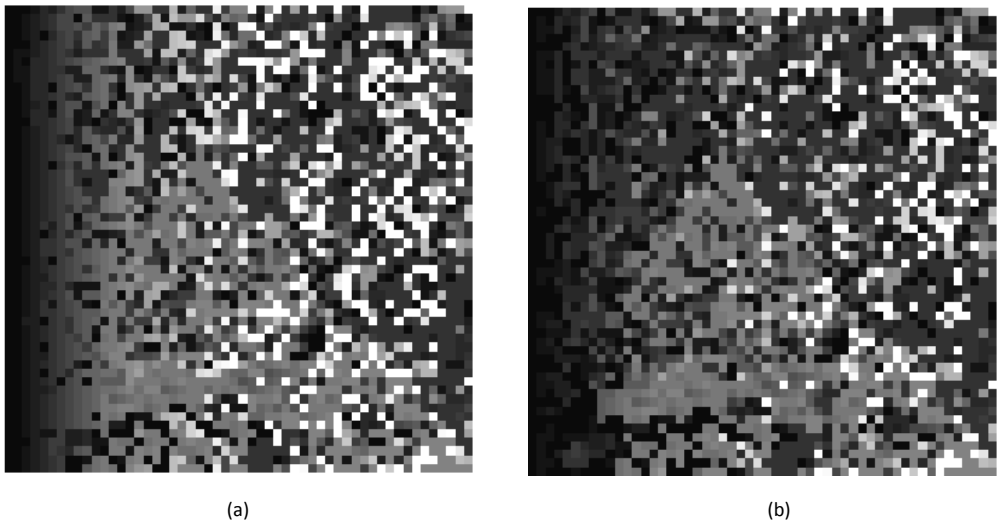
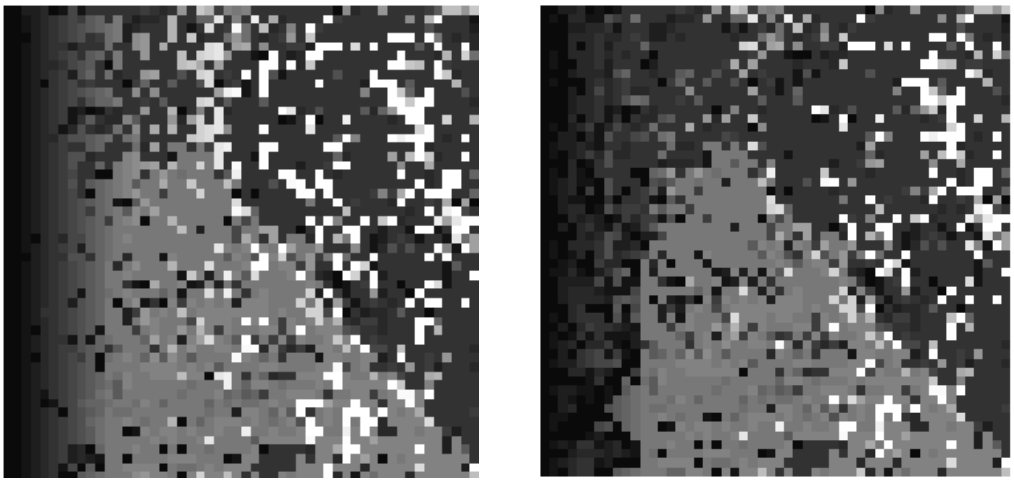


Figura 20 – Mapa de disparidade para arquitetura em (a)*hardware* (b)*software* para uma máscara 3x3

Para uma implementação em *hardware* de uma máscara 5x5 sobre uma imagem 56x56 obteve-se uma latência inical de 466 pulsos de relógio, após esta latência inicial, um píxel de saída será processado por cada ciclo de relógio, ou seja, mais 2704 pulsos de relógio para finalizar a imagem, totalizando uma latência total de 3150 pulsos de relógio. Sabe-se através da tabela 2 que a frequência de operação para esta arquitetura é de 195Mhz, portanto, o tempo total de execução é de 16,31us. Para uma implementação em *software* da mesma máscara e imagem obteve-se um tempo de execução de aproximadamente 3,822ms, portanto a arquitetura de *hardware* teve um fator de aceleração de 234 vezes. Os mapas de disparidade para a arquitetura de *hardware* e *software* citadas podem ser vistas nas figuras 21 (a) e (b) respectivamente.



(a) (b)

Figura 21 – Mapa de disparidade para arquitetura em (a)*hardware* (b)*software* para uma máscara 5x5

4.2.3 Comparação Númerica

As figuras 22 e 23 apresentam o mapa de disparidade obtido pela arquitetura de *hardware* e o mapa de disparidade apresentado na base de dados Middlebury, respectivamente. Os resultados foram obtidos para uma pequena porção da imagem original usando uma máscara de 7x7. A disparidade fornecida na base de dados é mostrada na Figura 31.

5	5	5	5	5	5	2	5	5
5	5	5	5	17	5	5	5	26
5	6	5	5	5	5	5	5	5
5	13	5	22	5	5	5	5	5
5	20	5	22	5	5	5	5	5
5	20	5	22	23	5	5	5	5
19	20	5	5	5	5	5	5	5
5	12	21	22	5	5	5	5	5
12	12	12	22	5	5	5	5	5
12	20	12	22	5	13	5	5	5
12	12	12	22	18	5	5	5	5
19	12	12	12	19	12	25	5	5
12	12	12	12	12	12	5	12	5
12	12	12	12	12	12	5	12	5
12	12	12	12	12	24	22	10	5
12	12	12	12	1	12	12	14	11
12	20	12	12	12	24	12	12	16
12	12	12	12	12	12	18	12	5
12	12	12	12	13	12	12	13	13
12	12	12	12	1	13	12	12	5
12	12	12	12	12	9	10	12	26
12	13	3	12	15	13	9	20	12
2	3	13	3	12	13	13	12	10
12	12	21	12	23	12	13	12	13

Figura 22 – Figura com valores do mapa de disparidade para máscara 7x7

4	4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4	4
4	4	4	4	4	4	4	4	4
12	4	4	4	4	4	4	4	4
12	12	4	4	4	4	4	4	4
12	12	12	4	4	4	4	4	4
12	12	12	12	4	4	4	4	4
12	12	12	12	12	4	4	4	4
12	12	12	12	12	12	4	4	4
12	12	12	12	12	12	12	4	4
12	12	12	12	12	12	12	4	4
12	12	12	12	12	12	12	12	4
12	12	12	12	12	12	12	12	12
12	12	12	12	12	12	12	12	12
12	12	12	12	12	12	12	12	12
12	12	12	12	12	12	12	12	12
12	12	12	12	12	12	12	12	12
12	12	12	12	12	12	12	12	12
12	12	12	12	12	12	12	12	12
12	12	12	12	12	12	12	12	12

Figura 23 – Figura com valores do mapa de disparidade da Middlebury

Analisando os dois mapas de disparidade, nota-se uma semelhança entre o resultado obtido pela arquitetura de hardware e a disparidade apresentada pela base de dados Middlebury. Para efeitos de comparação numérica, a disparidade fornecida pela base de dados pode ser usada como referência permitindo realizar uma estimação estatística sobre a qualidade do resultado obtido pela arquitetura de hardware. O erro quadrático médio obtido foi de 0.38, o que demonstra a similitude entre as duas disparidades.

4.3 Resultados do tempo de Execução

Foi realizada uma implementação em linguagem C do algoritmo que calcula a disparidade através da transformada Census 3x3 e do cálculo da distância de Hamming para duas imagens 56x56 obtidas no através da base de dados do sítio da MiddleBury (MIDDLEBURY, 2001), esta mesma imagem foi utilizada para o código em VHDL. Através de uma análise de desempenho (em relação ao tempo de execução) notou-se que o tempo de execução não superou a escala mínima da análise de *profile*, portanto, para se conseguir uma melhor análise, fez-se um *loop* para repetir todos procedimentos dez mil vezes. Os resultados dessa análise estão contidos na figura abaixo.

Pode-se concluir através da análise de *profile* que mais de 85% do tempo da execução encontra-se no cálculo da distância de Hamming. É possível observar que grande parte do tempo, equivalente a 6% do total, encontra-se na transformada Census, enquanto que o processamento da função *main*() utiliza aproximadamente 7% do tempo total. Isto pode ser explicado considerando os processos de controle e alocação de memória. É importante

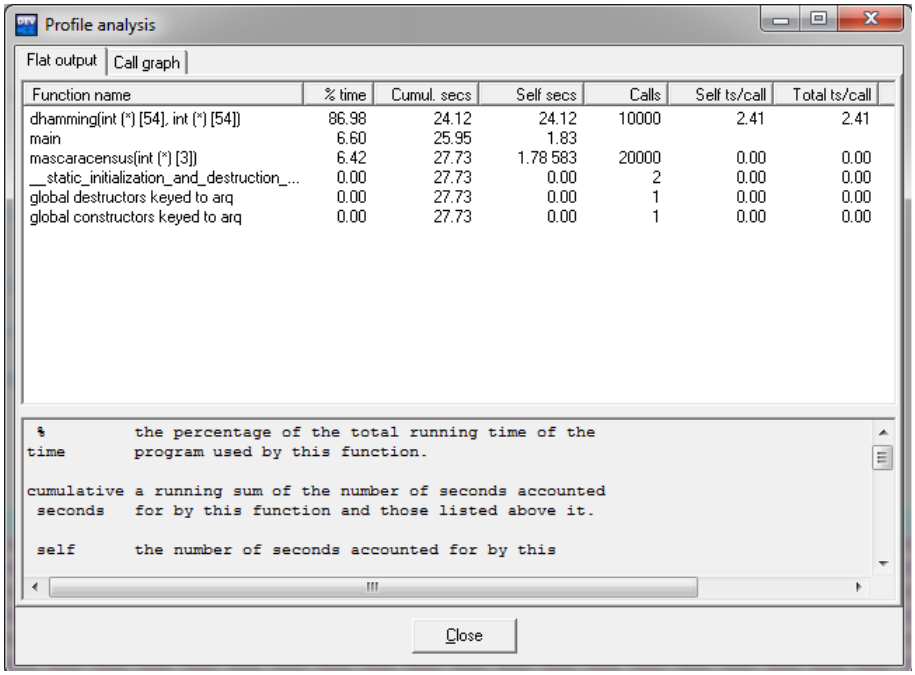


Figura 24 – Análise de *profile* do código em C

salientar que em uma implementação de *hardware* os processos de controle e alocação de memoria, geralmente, são realizados de forma simultânea.

O algoritmo implementado é altamente paralelizável dado que em *hardware* todas as comparações de uma máscara podem ser feitas simultaneamente enquanto que na implementação *software* as comparações são realizadas sequencialmente. Isto mostra que uma vantagem da utilização de um *hardware* dedicado em comparação com uma implementação em *software* seria o aceleramento do tempo de execução. Outro ponto de relevância é que em uma implementação em *hardware* várias máscaras podem ser processadas simultaneamente.

5 Discussão dos Resultados

Através dos resultados obtidos nas simulações é possível confirmar que o aumento de tamanho da máscara gera uma melhora na matriz de disparidade, entretanto gera consequências como o aumento da área de silício e a diminuição da frequência de operação do circuito.

Neste projeto foi adotado como limitante para os resultados uma frequência mínima de 180MHz, visto que é a frequência de um projeto feito pelos engenheiros da Xilinx na qual, um par estereoscópico para capturar as imagens e mostrar na saída HDMI, é utilizado, além disso, outro fator limitante é a quantidade de *slices* existentes na Spartan-6 que foi a FPGA utilizada neste projeto. As máscaras superiores 7x7 resultaram em um consumo superiror a 100% da FPGA e a arquitetura que se destacou foi com a máscara 7x7, visto que obteve uma frequência de operação superior a 180MHz e enquadrou-se nas limitações da FPGA.

Nota-se que com utilização da transformada Census uma parte da informação sobre a imagem é perdida o que acarreta na diminuição da precisão da disparidade final. É possível observar na figura 25 que as primeiras colunas sempre possuem disparidade baixa, isto deve-se ao fato de que a comparação é feita da imagem esquerda com a imagem direita, assim, considerando uma linha epipolar, um ponto na imagem esquerda pode estar na mesma coordenada da imagem direita ou em uma coordenada à sua esquerda, este fato é representado pela figura 26.

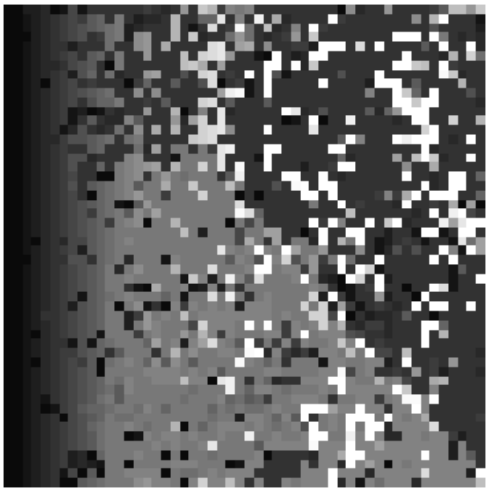


Figura 25 – Mapa de disparidade de uma arquitetura 5x5.

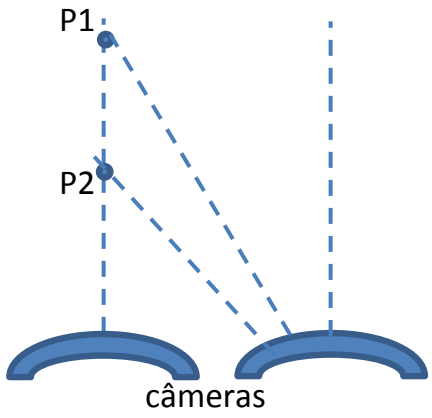


Figura 26 – Demonstração da localização dos pontos da imagem esquerda em relação à imagem direita.

A figura 27 apresenta outro ponto importante a ser ressaltado sobre a transformada Census em que se o valor central da janela for próximo do mínimo ou do máximo, o resultado desta transformada não representará com exatidão a intensidade dos pixels a sua volta, porém, como vistos nas referências é possível realizar modificações e aperfeiçoamentos sobre a transformada Census (FIFE, 2013) (AMBROSCH, 2009) (CHANG, 2010), de forma facilmente aplicável na arquitetura desenvolvida.

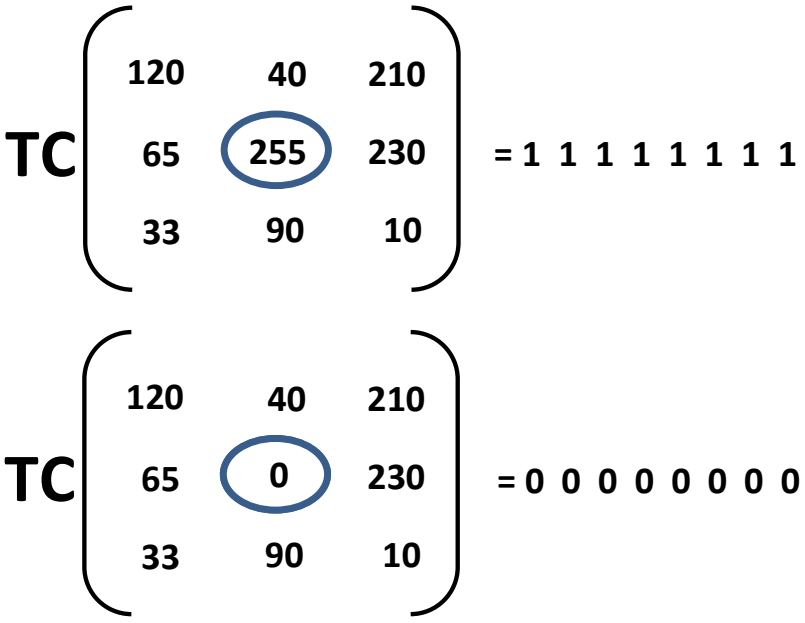


Figura 27 – Demonstração dos resultados da transformada Census para valores extremos.

Uma modificação possível sobre a transformada Census mostrada na figura 28 é a utilização da média de todos os valores da janela ao invés do valor central (NISHIMURA,

2008), isto fará com que valores extremos na posição central não causem tanta perda da informação da intensidade do pixel ao seu redor, neste caso a comparação da média também é feita com o pixel central, porém, para realizar esta modificação é necessário realizar divisões, o que acarreta em um grande aumento no uso de recursos de *hardware*.

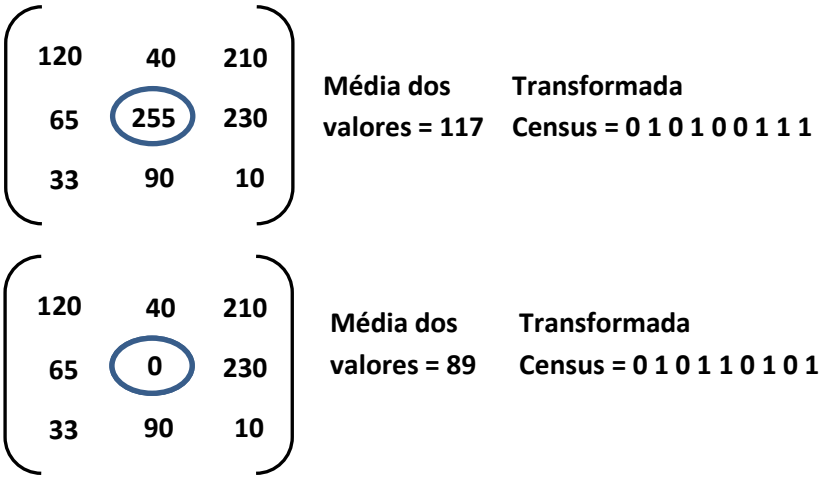


Figura 28 – Demonstração dos resultados da transformada Census para média dos valores.

Para contornar este problema da divisão foi proposto realizar a soma de apenas quatro valores do entorno do pixel central e fazer divisões por quatro, que é uma operação facilmente realizada com apenas dois deslocamentos, este processo é exemplificado na figura 29.

Figura 29 – Demonstração dos resultados da transformada Census para média dos quatro valores ao entorno do valor central.

A figura 30 mostra os resultados da implementação em *software* desta TC modificada proposta para máscaras (a)3x3 e (b)5x5.

Através de comparações qualitativas, observando as figuras 19, 30 e comparando com a disparidade real representada pelo mapa de disparidade figura 31 que possui um fator de multiplicação por oito, pode-se dizer que a transformada Census modificada proposta trouxe resultados superiores se comparado com a transforamda Census básica.

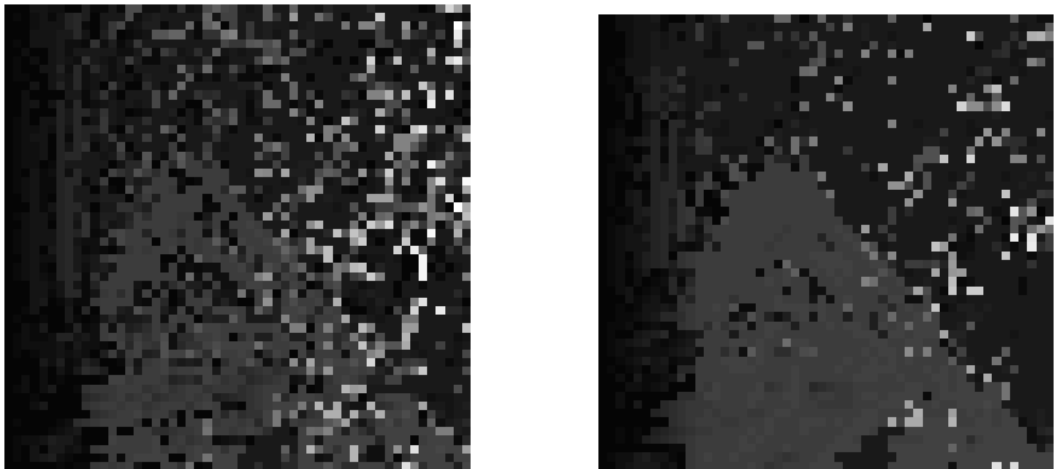


Figura 30 – (A)Mapa de disparidade para TC modificada 3x3 (B)Mapa de disparidade para TC modificada 5x5.

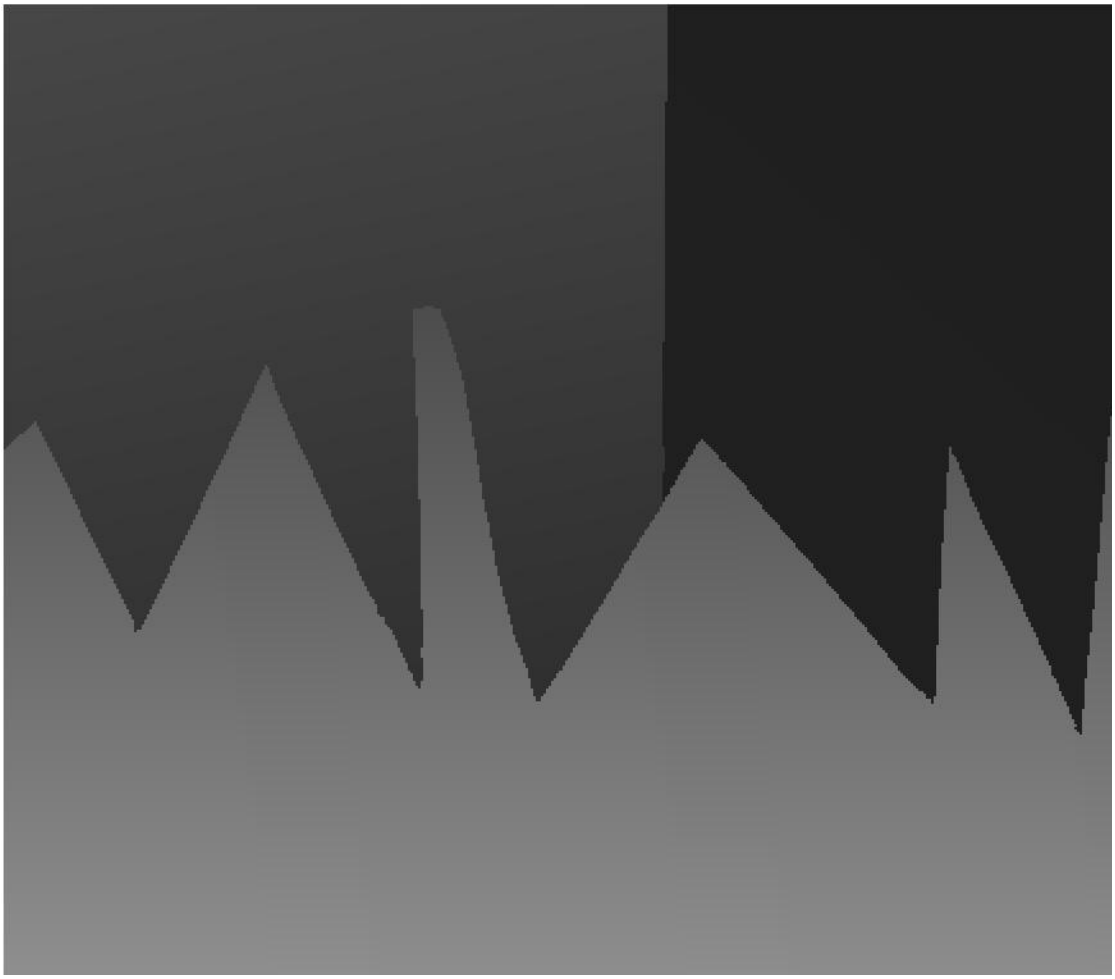


Figura 31 – Mapa de disparidade real do bando de dados Middlebury (MIDDLEBURY, 2001)

6 Conclusão

Para processamento espacial de imagens os dispositivos FPGAs são uma solução adequada, pois permite implementar arquiteturas de *hardware* dedicadas para o processamento paralelo. O trabalho foi desenvolvido seguindo a metodologia tradicional para projeto de circuitos digitiais em FPGAs e as arquiteturas de *hardware* foram descritas usando a linguagem VHDL.

Foi desenvolvido uma arquitetura de *hardware* dedicada para o cálculo do mapa de disparidade entre duas imagens adquiridas através de um sistema de visão estéreo de forma que permita calcular a distância frontal entre o deficiente visual e os obstáculos. Adicionalmente, foi desenvolvida uma ferramenta de geração automática de código VHDL capaz de gerar códigos para diferentes tamanhos de imagens e para diferentes tamanhos de máscaras de pixels. Esta ferramenta permitiu acelerar o tempo de desenvolvimento das implementações. Assim, as arquiteturas para máscaras 3x3, 5x5, 7x7, 9x9 e 11x11 foram implementada.

Os circuitos foram eficiente mapeados em FPGAs comerciais. Para o dispositivo escolhido (Spartan6 xc6slx45-3csg324), foi possível implementar máscaras de 3x3, 5x5, 7x7 e 9x9. Máscaras de tamanho maior requerem de um número maior de registradores.

Nas comparações entre *hardware* e *software* observou-se que o fator de aceleração para arquitetura de *hardware* superou 200 vezes a arquitetura de *software* o que permitiria o cálculo de disparidade em tempo real para uma grande quantidade de aplicações em engenharia.

Com base nos resultados de comparação entre os mapas de disparidade obtidos pelo *hardware* e *software* é possível concluir que as arquiteturas propostas apresentam resultados eficientes em termos da qualidade do mapa de disparidade.

É importante ressaltar que a utilização da transformada Census básica acrescenta ruído no processo de cálculo de correspondência entre as imagens, especificamente quando os objetos da imagem não possuem bordas bem definidas.

Considerando as capacidades de processamento em tempo real, obtidas pela implementação em *hardware*, assim como os métodos de fácil implementação para melhoria da qualidade do resultado, são considerados os seguintes trabalhos futuros:

1 Implementação em *hardware* da transformada Census modificada [(NISHIMURA, 2008)]

2 Integração do kit estereoscópico para validação das arquiteturas.

3 Implementação de um processador de *software* embarcado na FPGA para realizar o cálculo da equação 3.21 visando estimar a distância frontal aos obstáculos.

4 Melhoria na arquitetura da máscara buffer (vide figura 11) com intuito de reduzir o consumo de recursos. Para isto seria necessário acrescentar uma borda na imagem final para que esta tenha o mesmo tamanho da imagem original.

5 Cálculo da média móvel para diminuir ruído do mapa de disparidade.

Referências

AMBROSCH, K. Algorithmic considerations for real-time stereo vision applications. 2009. Citado na página 48.

BAILEY, D. G. *Design for Embedded image processing on FPGAs*. [S.l.]: John Wiley & Sons (Asia) Pte Ltd, 2011. Citado na página 13.

CALAZANS, N. *Projeto Lógico Automatizado de Sistemas Digitais Sequenciais*. [S.l.]: Imprinta Gráfica e Editora Ltda, 1998. Citado 2 vezes nas páginas 6 e 27.

CHANG, N. Y.-C. Algorith and architecture of disparity estimation with mini-census adaptive support weight. 2010. Citado na página 48.

COLODRO, C. Evaluación de algoritmos de correspondencia estereoscópica y su implementación en fpga. 2010. Citado 2 vezes nas páginas 23 e 30.

ESTEVES, G. Estereoscopia no cálculo de distância e controle de plataforma robótica. *Universidade de Pernambuco*, 2012. Citado 4 vezes nas páginas 6, 18, 20 e 22.

FIFE, W. S. Improved census transform for resource-optimized stereo vision. 2013. Citado na página 48.

FILHO, O. M. *Processamento Digital de Imagens*. [S.l.]: BRASPORT, 1999. Citado na página 22.

FRANÇA, J. A. de. *Desenvolvimento de Algoritmos de Visão Estereoscópica para Aplicações em Robótica Móvel*. Tese (Doutorado) — Universidade de Santa Catarina, 2003. Citado na página 24.

GAUTAMA, S. Evaluation of stereo matching algorihms for occupant detection. *IEEE*, 1999. Citado na página 30.

GONZALES, R. C. *Digital Image Processing*. [S.l.]: Tom Robbins, 2002. Citado 4 vezes nas páginas 6, 16, 17 e 22.

GONZALES, R. C. *Digital Image Processing using MATLAB*. [S.l.]: Gatesmark, 2009. Citado na página 16.

HAMZA, B. Fpga design of a real-time obstacle detection system using setereovision. *International Conference on Microelectronics (ICM)*, 2012. Citado na página 31.

IBARRA, M. Stereo vision algorithm implementation in fpga usiing census transform for effective resource optimization. *Euromicro Conference on Digital System Design*, 2009. Citado na página 30.

KUHN, M. Efficient asic implementation of a real-time depth mapping stereo vision system. *IEEE International Midwest Symposium on Circuits and Systems*, v. 3, 2003. Citado na página 30.

MANUAL de Legislação em Saúde da Pessoa com Deficiência. [S.l.], 2006. Minisério da Saúde. Citado na página 12.

MARTÍNEZ, M. M. *Técnicas de visión estereoscópica para determinar la estructura tridimensional de la escena*. Dissertação (Mestrado) — Universidad Coomplutense de Madrid, 2010. Citado 4 vezes nas páginas 6, 18, 22 e 24.

MEYER-BAESE. *Digital Signal Processing with Field Programmable Gate Arrays*. [S.l.]: Springer-Verlag, 2004. Citado na página 13.

MIDDLEBURY. http://vision.middlebury.edu/stereo/. Acessado em: 11/2013. 2001. Citado 4 vezes nas páginas 7, 40, 45 e 50.

MUNOZ, D. *Otimização por Inteligência de Enxames usando Arquiteturas para Aplicações Embarcadas*. Tese (Doutorado) — Universidade de BrasÍlia, 2012. Citado na página 28.

NAOULOU, A. Alternative to sequential architetures to improve the processing time of passive stereovision algorithms. *IEEE International Conference on Field Programmable Logic and Application*, 2006. Citado na página 30.

NISHIMURA, C. Análise comparativa de algoritmos de correlaÇão local baseados em intensidade luminosa. 2008. Citado 2 vezes nas páginas 49 e 51.

PEDRONI, V. A. *Circuit Desing with VHDL*. [S.l.]: Massachusetts Institute of Technology, 2004. Citado na página 28.

QIU, L. L. L. Contour extraction of moving objects. *IEEE International Conference on Patter Recognition*, v. 2, 1998. Citado na página 29.

SASS, A. G. S. R. *Embedded Systems Design with Platform FPGA, Principles and Practices*. [S.l.]: Marganm Kaufmann, 2010. Citado na página 13.

SMITH, M. *Application-Specific Integrated Circuits*. [S.l.]: USA: Addion-Wesley, 1997. Citado na página 27.

WOODFILL, J. Frame-rate robust stereo on a pci board. 1999. Citado na página 30.

WOODFILL, J. Tyzx deepsea high speed stereo vision system. *IEEE Conferece on Computer vision and PAttern Recognition*, 2004. Citado na página 30.

WU, Y. T. K. Structured asic, evolution or revolution. *International Symposium on Physical Design*, 2004. Citado na página 27.

YAMADA, T. I. K. Generation of a disparity panorama using a 3-camera capturing system. *IEEE Internetional Conference on Image Processing*, v. 2, 2000. Citado na página 30.

Anexos

ANEXO A – *Software* para o cálculo da disparidade baseada na transformada Census

```
#include<stdio.h>
#include<stdlib.h>
#include<iostream>
#include<windows.h>

#define DM 3
#define ML 56
#define MC 56
FILE *arq;
int mascaracensus(int matriz[DM][DM]){
    int tcensus=0,i,j,x;
//  x=8388608;//5x5
    x=128;//3x3
    for(i=0;i<DM;i++)for(j=0;j<DM;j++)if(i!=DM/2 || j!=DM/2){
        if(matriz[i][j]>matriz[DM/2][DM/2])tcensus=tcensus ^ x;
        x/=2;
        }
  return tcensus;
}
int dhamming(int tcensus2[ML-DM+1][MC-DM+1], int tcensus3[ML-DM+1][MC-DM+1]){
    int aux3=0,aux2=0,aux=0;
    int num=128;
    int val[ML-DM+1][MC-DM+1]={},maior[ML-DM+1][MC-DM+1]={};
    int i,j,l;
for(i=0;i<=ML-DM;i++)for(j=0;j<=MC-DM;j++)for(l=0;l<=j;l++){
        aux2=tcensus2[i][j];
        aux3=tcensus3[i][l];
        do{
           if(aux2>=num && aux3>=num)aux++;
           if(aux2<num && aux3<num)aux++;
           if(aux2>=num)aux2-=num;
           if(aux3>=num)aux3-=num;
           num/=2;
        }while(num>0);//final do while
        if(aux>=val[i][j]){
            val[i][j]=aux;
            maior[i][j]=j-l+1;
            }
        aux=0;num=128;
    }//final do for l

    arq = fopen("tccteste2.txt","w");
        for(i=0;i<=MC-DM;i++){
            for(j=0;j<=MC-DM;j++)
                fprintf(arq,"%d\t",maior[i][j]);
            fprintf(arq,"\n");
            }
    fclose(arq);

}//final da dhamming

int main(){

int matriz2[ML][MC]=
```

```
{
    //Imagem da esquerda
};

int matriz3[ML][MC]=
{
    //Imagem da direita
};

int inicio=0,final=0,tmili=0;
int matriz[DM][DM];
int tcensus2[ML-DM+1][MC-DM+1]={};
int tcensus3[ML-DM+1][MC-DM+1]={};
int i,j,h,k,t1,t2,count,teste;
inicio = GetTickCount();
for(k=0;k<(ML-DM+1);k++)for(h=0;h<(MC-DM+1);h++){

    for(i=0;i<DM;i++)for(j=0;j<DM;j++)matriz[i][j]=matriz2[i+k][j+h];
    tcensus2[k][h]=mascaracensus(matriz);

    for(i=0;i<DM;i++)for(j=0;j<DM;j++)matriz[i][j]=matriz3[i+k][j+h];
    tcensus3[k][h]=mascaracensus(matriz);
}
dhamming(tcensus2,tcensus3);
    final = GetTickCount();
    tmili = final - inicio;

    printf("tempo decorrido: %d\n", tmili);

system("pause");
return 0;
}
```

ANEXO B – Gerador automático de código em VHDL para o cálculo da disparidade baseada na transformada Census

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#define NUMCOLUNAS 56
#define TotalJanela 25

int main(){

int t_mascara=(int)sqrt(TotalJanela);
int disparidade_maxima=NUMCOLUNAS-t_mascara+1;
int numero_bits_disp=int(log2(disparidade_maxima));
int numero_bits_dh=int(log2(TotalJanela-1)+1);

FILE *arq;
arq = fopen("Debugging.vhd" , "w");
fprintf(arq,"library IEEE;\n");
fprintf(arq,"use IEEE.STD_LOGIC_1164.ALL;\n");
fprintf(arq,"\n");
fprintf(arq,"entity Debugging is\n");
fprintf(arq,"    Port (     clk : in  STD_LOGIC;\n");
fprintf(arq,"               readyDTC: out std_logic;\n");
fprintf(arq,"               doneROM: out std_logic;\n");
fprintf(arq,"               s_DISPfinal: out std_logic_vector(%d downto 0);\n",
numero_bits_disp);
fprintf(arq,"               donedh: out std_logic_vector(%d downto 0);--sinal
desnecessário\n",disparidade_maxima-1);
fprintf(arq,"               reset : in  STD_LOGIC\n");
fprintf(arq,"               );\n");
fprintf(arq,"end Debugging;\n");
fprintf(arq,"\n");
fprintf(arq,"architecture Behavioral of Debugging is\n");
fprintf(arq,"signal s_saidaD : std_logic_vector (7 downto 0);\n");
fprintf(arq,"signal s_saidaE : std_logic_vector (7 downto 0);\n");
fprintf(arq,"signal s_readyE: std_logic;\n");
fprintf(arq,"signal s_readyD: std_logic;\n");
fprintf(arq,"signal s_readyDTC: std_logic;\n");
fprintf(arq,"signal s_saida_tc_E : std_logic_vector(%d DOWNTO 0);\n",TotalJanela*8-1);
fprintf(arq,"signal s_saida_tc_D : std_logic_vector(%d DOWNTO 0);\n",TotalJanela*8-1);
fprintf(arq,"signal s_readyTCE : std_logic;\n");
fprintf(arq,"signal s_readyTCD : std_logic;\n");
fprintf(arq,"signal s_resultadoE : std_logic_vector(%d downto 0);\n",TotalJanela-2);
fprintf(arq,"signal s_resultadoD : std_logic_vector(%d downto 0);\n",TotalJanela-2);
fprintf(arq,"signal s_DISPout : std_logic_vector (%d downto 0);\n",(numero_bits_disp+1)*
disparidade_maxima-1);
fprintf(arq,"signal s_TCEout : std_logic_vector(%d downto 0);\n",TotalJanela-2);
fprintf(arq,"signal s_TCDout : std_logic_vector (%d downto 0);\n",(TotalJanela-1)*
disparidade_maxima-1);
fprintf(arq,"signal s_DISPoutDH: std_logic_vector(%d downto 0);\n",(numero_bits_disp+1)*
disparidade_maxima-1);
fprintf(arq,"signal s_resultadoDH: std_logic_vector( %d downto 0);\n",numero_bits_dh*
disparidade_maxima-1);
fprintf(arq,"component AMROM\n");
fprintf(arq,"    Port (  clk : in STD_LOGIC;\n");
```

```
49  fprintf(arq,"                reset : in STD_LOGIC;\n");
50  fprintf(arq,"                done : out std_logic;\n");
51  fprintf(arq,"                saidaD : out std_logic_vector (7 downto 0);\n");
52  fprintf(arq,"                saidaE : out std_logic_vector (7 downto 0)\n");
53  fprintf(arq,"    );\n");
54  fprintf(arq,"end component;\n");
55  fprintf(arq,"component DDP\n");
56  fprintf(arq,"Port (         entrada: in std_logic_vector(7 DOWNTO 0);\n");
57  fprintf(arq,"               reset: in std_logic;\n");
58  fprintf(arq,"               clk: in std_logic;\n");
59  fprintf(arq,"               ready: out std_logic;\n");
60  fprintf(arq,"               saida : out std_logic_vector(%d DOWNTO 0));\n",TotalJanela*8
    -1);
61  fprintf(arq,"end component;\n");
62  fprintf(arq,"component transformadacensus is\n");
63  fprintf(arq,"    Port (  valor1: in std_logic_vector(%d DOWNTO 0);\n",TotalJanela*8-1);
64  fprintf(arq,"               clk: in STD_LOGIC;\n");
65  fprintf(arq,"               controle: in std_logic;\n");
66  fprintf(arq,"               ready: out std_logic;\n");
67  fprintf(arq,"               resultado: out std_logic_vector(%d downto 0));\n",
    TotalJanela-2);
68  fprintf(arq,"end component;\n");
69  fprintf(arq,"component DTC is\n");
70  fprintf(arq,"    Port (         TCDin : in std_logic_vector(%d downto 0);\n",TotalJanela
    -2);
71  fprintf(arq,"                   TCDout : out std_logic_vector (%d downto 0);\n",(
    TotalJanela-1)*disparidade_maxima-1);
72  fprintf(arq,"                  DISPout : out std_logic_vector (%d downto 0);\n",(
    numero_bits_disp+1)*disparidade_maxima-1);
73  fprintf(arq,"                   reset : in std_logic;\n");
74  fprintf(arq,"                   done : out std_logic;\n");
75  fprintf(arq,"                   start : in std_logic;\n");
76  fprintf(arq,"                   clk : in std_logic);\n");
77  fprintf(arq,"end component;\n");
78  fprintf(arq,"component Atraso_TCE is\n");
79  fprintf(arq,"Port (         entrada : in std_logic_vector(%d downto 0);\n",TotalJanela-2
    );
80  fprintf(arq,"               reset: in std_logic;\n");
81  fprintf(arq,"               clk: in std_logic;\n");
82  fprintf(arq,"               saida : out std_logic_vector(%d downto 0));\n",TotalJanela-2
    );
83  fprintf(arq,"end component;\n");
84  fprintf(arq,"component DH is\n");
85  fprintf(arq,"port(  vetor1: in std_logic_vector(%d downto 0);\n",TotalJanela-2);
86  fprintf(arq,"       vetor2: in std_logic_vector(%d downto 0);\n",TotalJanela-2);
87  fprintf(arq,"       DISPin: in std_logic_vector(%d downto 0);\n",numero_bits_disp);
88  fprintf(arq,"       DISPout: out std_logic_vector(%d downto 0);\n",numero_bits_disp);
89  fprintf(arq,"       start: in std_logic;\n");
90  fprintf(arq,"       done: out std_logic;\n");
91  fprintf(arq,"       clk: in std_logic;\n");
92  fprintf(arq,"       resultado: out std_logic_vector( %d downto 0));\n",numero_bits_dh-1);
93  fprintf(arq,"end component;\n");
94  fprintf(arq,"component TOTALcompDISP is\n");
95  fprintf(arq,"                   port(\n");
```

```
 96   fprintf(arq,"                    resultadoDH: in std_logic_vector( %d downto 0);\n",
      numero_bits_dh*disparidade_maxima-1);
 97   fprintf(arq,"                    DISPoutDH: in std_logic_vector(%d downto 0);\n",(
      numero_bits_disp+1)*disparidade_maxima-1);
 98   fprintf(arq,"                    DISPfinal: out std_logic_vector(%d downto 0);\n",
      numero_bits_disp);
 99   fprintf(arq,"                    clk: in std_logic\n");
100   fprintf(arq,"                    );\n");
101   fprintf(arq,"end component;\n");
102   fprintf(arq,"begin\n");
103   fprintf(arq,"readyDTC <= s_ReadyDTC;\n");
104   fprintf(arq,"compAMROM : AMROM\n");
105   fprintf(arq,"  PORT MAP (\n");
106   fprintf(arq,"    clk => clk,\n");
107   fprintf(arq,"    reset => reset,\n");
108   fprintf(arq,"    done => doneROM,\n");
109   fprintf(arq,"    saidaD => s_saidaD,\n");
110   fprintf(arq,"    saidaE => s_saidaE\n");
111   fprintf(arq,"  );\n");
112   fprintf(arq,"DDP_Direita: DDP\n");
113   fprintf(arq,"Port map( entrada => s_saidaD,\n");
114   fprintf(arq,"            reset => reset,\n");
115   fprintf(arq,"            clk => clk,\n");
116   fprintf(arq,"            ready => s_readyD,\n");
117   fprintf(arq,"            saida => s_saida_tc_D\n");
118   fprintf(arq,");\n");
119   fprintf(arq,"DDP_Esquerda: DDP\n");
120   fprintf(arq,"Port map( entrada => s_saidaE,\n");
121   fprintf(arq,"            reset => reset,\n");
122   fprintf(arq,"            clk => clk,\n");
123   fprintf(arq,"            ready => s_readyE,\n");
124   fprintf(arq,"            saida => s_saida_tc_E\n");
125   fprintf(arq,");\n");
126   fprintf(arq,"TCD:transformadacensus\n");
127   fprintf(arq,"Port map(   valor1 => s_saida_tc_D,\n");
128   fprintf(arq,"               clk =>clk,\n");
129   fprintf(arq,"               controle =>s_readyD,\n");
130   fprintf(arq,"               ready =>s_readyTCD,\n");
131   fprintf(arq,"               resultado =>s_resultadoD);\n");
132   fprintf(arq,"\n");
133   fprintf(arq,"TCE:transformadacensus\n");
134   fprintf(arq,"Port map(   valor1 => s_saida_tc_E,\n");
135   fprintf(arq,"               clk =>clk,\n");
136   fprintf(arq,"               controle =>s_readyE,\n");
137   fprintf(arq,"               ready =>s_readyTCE,\n");
138   fprintf(arq,"               resultado =>s_resultadoE);\n");
139   fprintf(arq,"DeslTransCensusDireita:DTC\n");
140   fprintf(arq,"Port map(      TCDin => s_resultadoD,\n");
141   fprintf(arq,"                  TCDout  =>s_TCDout,\n");
142   fprintf(arq,"                 DISPout =>s_DISPout,\n");
143   fprintf(arq,"                  done => s_readyDTC,\n");
144   fprintf(arq,"                  reset => reset,\n");
145   fprintf(arq,"                  start => s_readyTCD,\n");
146   fprintf(arq,"                  clk => clk);\n");
```

```
147   fprintf(arq,"AtrasoTCE:Atraso_TCE\n");
148   fprintf(arq,"Port map(  entrada => s_resultadoE,\n");
149   fprintf(arq,"               reset => reset,\n");
150   fprintf(arq,"               clk => clk,\n");
151   fprintf(arq,"               saida =>s_TCEout);\n");
152   fprintf(arq,"gen_dh:\n");
153   fprintf(arq,"   for n in 1 to %d generate\n",disparidade_maxima);
154   fprintf(arq,"       calcDH:DH\n");
155   fprintf(arq,"       Port map(           vetor1 => s_TCEout,\n");
156   fprintf(arq,"                                   vetor2 => s_TCDout(%d+%d*(n-1) downto
      0+%d*(n-1)),\n",TotalJanela-2,TotalJanela-1,TotalJanela-1);
157   fprintf(arq,"                                   DISPin =>s_DISPout(%d+%d*(n-1) downto
      0+%d*(n-1)),\n",numero_bits_disp,numero_bits_disp+1,numero_bits_disp+1);
158   fprintf(arq,"                                   DISPout =>s_DISPoutDH(%d+%d*(n-1) downto
      0+%d*(n-1)),\n",numero_bits_disp,numero_bits_disp+1,numero_bits_disp+1);
159   fprintf(arq,"                                   start=> s_readyDTC,\n");
160   fprintf(arq,"                                   done=>doneDH(n-1),\n");
161   fprintf(arq,"                                   clk => clk,\n");
162   fprintf(arq,"                                   resultado =>s_resultadoDH(%d+%d*(n-1)
      downto 0+%d*(n-1))\n",numero_bits_dh-1,numero_bits_dh,numero_bits_dh);
163   fprintf(arq,"                                   );\n");
164   fprintf(arq,"end generate gen_dh;\n");
165   fprintf(arq,"Minimi_DISP:TOTALcompDISP\n");
166   fprintf(arq,"                   port map(\n");
167   fprintf(arq,"                   resultadoDH =>s_resultadoDH,\n");
168   fprintf(arq,"                   DISPoutDH =>s_DISPoutDH,\n");
169   fprintf(arq,"                   DISPfinal =>s_DISPfinal,\n");
170   fprintf(arq,"                   clk => clk\n");
171   fprintf(arq,"                   );\n");
172   fprintf(arq,"\n");
173   fprintf(arq,"end Behavioral;\n");
174   fclose(arq);
175
176   arq = fopen("AMROM.vhd" , "w");
177   fprintf(arq,"library IEEE;\n");
178   fprintf(arq,"use ieee.std_logic_1164.all;\n");
179   fprintf(arq,"use ieee.std_logic_unsigned.all;\n");
180   fprintf(arq,"\n");
181   fprintf(arq,"entity AMROM is\n");
182   fprintf(arq,"    Port ( \n");
183   fprintf(arq,"    clk : in STD_LOGIC;\n");
184   fprintf(arq,"    reset : in STD_LOGIC;\n");
185   fprintf(arq,"    done : out std_logic;\n");
186   fprintf(arq,"    saidaD : out  std_logic_vector (7 downto 0);\n");
187   fprintf(arq,"    saidaE : out  std_logic_vector (7 downto 0)\n");
188   fprintf(arq,"    );\n");
189   fprintf(arq,"end AMROM;\n");
190   fprintf(arq,"\n");
191   fprintf(arq,"architecture Behavioral of AMROM is\n");
192   fprintf(arq,"\n");
193   int tam_img=NUMCOLUNAS*NUMCOLUNAS-1;
194   int qtd_bits_ende=int(log2(tam_img));
195   char ende_max_bin[qtd_bits_ende+1];
196   itoa(tam_img,ende_max_bin,2);
```

```
197   fprintf(arq,"constant addrs_max: std_logic_vector(%d downto 0) := \"%s\";\n",
      qtd_bits_ende,ende_max_bin);
198   fprintf(arq,"signal count: std_logic_vector(%d downto 0):= (others => '0');\n",
      qtd_bits_ende);
199   fprintf(arq,"\n");
200   fprintf(arq,"component ImgD\n");
201   fprintf(arq,"  PORT (\n");
202   fprintf(arq,"    clka : in std_logic;\n");
203   fprintf(arq,"    addra : in std_logic_vector(%d downto 0);\n",qtd_bits_ende);
204   fprintf(arq,"    douta : out std_logic_vector(7 downto 0)\n");
205   fprintf(arq,"    );\n");
206   fprintf(arq,"end component;\n");
207   fprintf(arq,"\n");
208   fprintf(arq,"component ImgE\n");
209   fprintf(arq,"  port (\n");
210   fprintf(arq,"    clka : in std_logic;\n");
211   fprintf(arq,"    addra : in std_logic_vector(%d downto 0);\n",qtd_bits_ende);
212   fprintf(arq,"    douta : out std_logic_vector(7 downto 0)\n");
213   fprintf(arq,"  );\n");
214   fprintf(arq,"end component;\n");
215   fprintf(arq,"\n");
216   fprintf(arq,"begin\n");
217   fprintf(arq,"\n");
218   fprintf(arq,"ROMD : ImgD\n");
219   fprintf(arq,"  PORT MAP (\n");
220   fprintf(arq,"    clka => clk,\n");
221   fprintf(arq,"    addra => count,\n");
222   fprintf(arq,"    douta => saidaD\n");
223   fprintf(arq,"  );\n");
224   fprintf(arq,"ROME : ImgE\n");
225   fprintf(arq,"  PORT MAP (\n");
226   fprintf(arq,"    clka => clk,\n");
227   fprintf(arq,"    addra => count,\n");
228   fprintf(arq,"    douta => saidaE\n");
229   fprintf(arq,"  );\n");
230   fprintf(arq,"process (reset,clk)\n");
231   fprintf(arq,"begin\n");
232   fprintf(arq,"if reset = '0' then\n");
233   fprintf(arq,"         count <= (others => '0');\n");
234   fprintf(arq,"           done <= '0';\n");
235   fprintf(arq,"     elsif rising_edge(clk) then\n");
236   fprintf(arq,"         if count = addrs_max then\n");
237   fprintf(arq,"             done <= '1';\n");
238   fprintf(arq,"         else\n");
239   fprintf(arq,"               done <= '0';\n");
240   fprintf(arq,"             count <= count + 1;\n");
241   fprintf(arq,"         end if;\n");
242   fprintf(arq,"     end if;\n");
243   fprintf(arq,"end process;\n");
244   fprintf(arq,"\n");
245   fprintf(arq,"end Behavioral;\n");
246   fprintf(arq,"\n");
247   fprintf(arq,"\n");
248   fclose(arq);
```

```
249
250   arq = fopen("DDP.vhd" , "w");
251   fprintf(arq,"library IEEE;\n");
252   fprintf(arq,"use IEEE.STD_LOGIC_1164.ALL;\n");
253   fprintf(arq,"USE ieee.std_logic_unsigned.all;\n");
254   fprintf(arq,"\n");
255   fprintf(arq,"entity DDP is\n");
256   fprintf(arq,"       Port (entrada : in  std_logic_vector(7 DOWNTO 0);\n");
257   fprintf(arq,"               reset: in std_logic;\n");
258   fprintf(arq,"               clk: in std_logic;\n");
259   fprintf(arq,"               ready: out std_logic;\n");
260   fprintf(arq,"               saida : out std_logic_vector(%d DOWNTO 0)\n",TotalJanela*8-1
      );
261   fprintf(arq,"               );\n");
262   fprintf(arq,"end DDP;\n");
263   fprintf(arq,"\n");
264   fprintf(arq,"architecture Behavioral of DDP is\n");
265   fprintf(arq,"\n");
266   int tam_buffer = NUMCOLUNAS*(t_mascara-1)+t_mascara+(disparidade_maxima-1)*(t_mascara-1);
267   int qtd_bits_buffer=int(log2(tam_buffer))+1;
268   char tam_buffer_bin[qtd_bits_buffer];
269   itoa(tam_buffer,tam_buffer_bin,2);
270   fprintf(arq,"constant constante : std_logic_vector(%d downto 0):=\"%s\";--13+4 2 linhas
      +3 pixels + 4pixels(pulos)\n",qtd_bits_buffer-1,tam_buffer_bin);
271   int tam_2 = NUMCOLUNAS-t_mascara+1;
272   int qtd_bits_tam2=int(log2(tam_2))+1;
273   char tam_2_bin[qtd_bits_tam2];
274   itoa(tam_2,tam_2_bin,2);
275   fprintf(arq,"constant constante2 : std_logic_vector(%d downto 0):=\"%s\";--numero de
      colunas - (tamanho da mascara - 1)\n",qtd_bits_tam2-1,tam_2_bin);
276   fprintf(arq,"signal s_ready : std_logic:='0';\n");
277   fprintf(arq,"signal s_ligacao : std_logic_vector(%d downto 0):=(others => '0');--(13+4
      2 linhas +3 pixels + 4pixels(pulos) +1 )*8 -1\n",tam_buffer*8+7);
278   fprintf(arq,"signal count : std_logic_vector(%d downto 0);--vai ate o valor da
      constante\n",qtd_bits_buffer-1);
279   fprintf(arq,"signal count2 : std_logic_vector(%d downto 0);--vai ate o valor da
      constante2\n",qtd_bits_tam2-1);
280   int qtd_bits_linhas=int(log2(NUMCOLUNAS-t_mascara+1));
281   fprintf(arq,"signal linha : std_logic_vector(%d downto 0);--informa em qual linha está
      para que possa pular\n",qtd_bits_linhas);
282   fprintf(arq,"\n");
283   fprintf(arq,"component ADP\n");
284   fprintf(arq,"   port(   entrada: in std_logic_vector(7 downto 0);\n");
285   fprintf(arq,"               reset: in std_logic;\n");
286   fprintf(arq,"               clk: in std_logic;\n");
287   fprintf(arq,"               saida: out std_logic_vector(7 downto 0) );\n");
288   fprintf(arq,"end component;\n");
289   fprintf(arq,"\n");
290   fprintf(arq,"begin\n");
291   fprintf(arq,"\n");
292   fprintf(arq,"s_ligacao(7 downto 0)<=entrada;\n");
293   fprintf(arq,"\n");
294   fprintf(arq,"gen_adp:\n");
295   fprintf(arq,"for n in 1 to %d generate --13+4 2 linhas +3 pixels + 4pixels(pulos)\n",
```

```
    tam_buffer);
296 fprintf(arq,"comp : ADP port map \n");
297 fprintf(arq,"               (entrada=>s_ligacao((n*8-1) downto (n-1)*8),\n");
298 fprintf(arq,"               reset=>reset,\n");
299 fprintf(arq,"               clk=>clk,\n");
300 fprintf(arq,"               saida=>s_ligacao(((n+1)*8-1) downto n*8)\n");
301 fprintf(arq,"               );\n");
302 fprintf(arq,"end generate gen_adp;\n");
303 fprintf(arq,"\n");
304 fprintf(arq,"process(clk,reset)--apenas informa quando os valores estão disponíveis
    para o cálculo\n");
305 fprintf(arq,"begin\n");
306 fprintf(arq,"   if reset = '0' then\n");
307 fprintf(arq,"       count<=(others => '0');\n");
308 fprintf(arq,"       s_ready<='0';\n");
309 fprintf(arq,"       ready <= '0';\n");
310 fprintf(arq,"   elsif rising_edge(clk) then\n");
311 fprintf(arq,"       ready <= s_ready;\n");
312 fprintf(arq,"       if (count=constante) then\n");
313 fprintf(arq,"           s_ready<='1';\n");
314 fprintf(arq,"       else\n");
315 fprintf(arq,"           s_ready<='0';\n");
316 fprintf(arq,"           count <= count +'1';\n");
317 fprintf(arq,"       end if;\n");
318 fprintf(arq,"   end if;\n");
319 fprintf(arq,"end process;\n");
320 fprintf(arq,"\n");
321 fprintf(arq,"process(clk,s_ready)\n");
322 fprintf(arq,"begin\n");
323 fprintf(arq,"   if s_ready = '0' then\n");
324 fprintf(arq,"       count2<=\"");
325 for(int i=0;i<=qtd_bits_tam2-1;i++){
326     if(i<qtd_bits_tam2-1)
327         fprintf(arq,"0");
328     else
329         fprintf(arq,"1");
330     }
331 fprintf(arq,"\";\n");
332 fprintf(arq,"       linha <= \"");
333 for(int i=0;i<=qtd_bits_linhas;i++)
334     fprintf(arq,"0");
335 fprintf(arq,"\";\n");
336 fprintf(arq,"   elsif rising_edge(clk) then\n");
337 fprintf(arq,"       if (count2=constante2) then\n");
338 fprintf(arq,"           linha <= linha + '1';\n");
339 fprintf(arq,"           count2 <= \"");
340 for(int i=0;i<=qtd_bits_tam2-1;i++){
341     if(i<qtd_bits_tam2-1)
342         fprintf(arq,"0");
343     else
344         fprintf(arq,"1");
345     }
346 fprintf(arq,"\";\n");
347 fprintf(arq,"       else\n");
```

```
348   fprintf(arq,"           count2 <= count2 +\"");
349   for(int i=0;i<=qtd_bits_tam2-1;i++){
350       if(i<qtd_bits_tam2-1)
351           fprintf(arq,"0");
352       else
353           fprintf(arq,"1");
354       }
355   fprintf(arq,"\";\n");
356   fprintf(arq,"       end if;\n");
357   fprintf(arq,"   end if;\n");
358   fprintf(arq,"end process;\n");
359   fprintf(arq,"\n");
360   fprintf(arq,"process(clk,reset,s_ready,linha)--processo para \"pulos\" a cada pulo (2)
      desloca 2 pixels a mascara\n");
361   fprintf(arq,"begin\n");
362   fprintf(arq,"   if reset = '0' or s_ready = '0' then\n");
363   fprintf(arq,"       saida<=(others => '0');\n");
364   fprintf(arq,"   elsif rising_edge(clk) then\n");
365   fprintf(arq,"       if linha = \"");
366   for(int i=0;i<=qtd_bits_linhas;i++)
367       fprintf(arq,"0");
368   fprintf(arq,"\" then\n");
369   char bla[qtd_bits_linhas];
370   for(int j=0;j<NUMCOLUNAS-t_mascara+1;j++){
371       fprintf(arq,"           saida<=");
372       for(int i=0;i<t_mascara;i++){
373           fprintf(arq,"s_ligacao(%d downto %d)",8*t_mascara+7+NUMCOLUNAS*8*(t_mascara-1-i
              )+(t_mascara-1)*8*(NUMCOLUNAS-t_mascara)-(t_mascara-1)*8*j,8+NUMCOLUNAS*8*(
              t_mascara-1-i)+(t_mascara-1)*8*(NUMCOLUNAS-t_mascara)-(t_mascara-1)*8*j);
374           if(i<t_mascara-1)
375               fprintf(arq," & ");
376           else
377               fprintf(arq,";\n");
378           }
379       itoa(j+1,bla,2);
380       int bla2=int(log2(j+1));
381       if(j<NUMCOLUNAS-t_mascara-1){
382           fprintf(arq,"       elsif linha = \"");
383           for(int k=bla2;k<qtd_bits_linhas;k++)
384               fprintf(arq,"0");
385           fprintf(arq,"%s",bla);
386           fprintf(arq,"\" then\n");
387       }
388       if(j==NUMCOLUNAS-t_mascara-1)
389           fprintf(arq,"       else\n");
390       }
391   fprintf(arq,"       end if;\n");
392   fprintf(arq,"   end if;\n");
393   fprintf(arq,"end process;\n");
394   fprintf(arq,"\n");
395   fprintf(arq,"\n");
396   fprintf(arq,"end Behavioral;\n");
397   fprintf(arq,"\n");
398   fprintf(arq,"\n");
```

```
399   fclose(arq);
400
401   arq = fopen("ADP.vhd" , "w");
402   fprintf(arq,"library IEEE;\n");
403   fprintf(arq,"use IEEE.STD_LOGIC_1164.ALL;\n");
404   fprintf(arq,"\n");
405   fprintf(arq,"entity ADP is\n");
406   fprintf(arq,"   port(   entrada: in std_logic_vector(7 downto 0);\n");
407   fprintf(arq,"               reset: in std_logic;\n");
408   fprintf(arq,"               clk: in std_logic;\n");
409   fprintf(arq,"               saida: out std_logic_vector(7 downto 0));\n");
410   fprintf(arq,"end ADP;\n");
411   fprintf(arq,"\n");
412   fprintf(arq,"architecture Behavioral of ADP is\n");
413   fprintf(arq,"signal s_saida : std_logic_vector(7 downto 0):=(others => '0');\n");
414   fprintf(arq,"begin\n");
415   fprintf(arq,"saida<=s_saida;\n");
416   fprintf(arq,"process(clk,reset)\n");
417   fprintf(arq,"begin\n");
418   fprintf(arq,"   if (reset = '0') then \n");
419   fprintf(arq,"       s_saida<=(others=>'0');\n");
420   fprintf(arq,"   elsif rising_edge(clk) then\n");
421   fprintf(arq,"       s_saida <= entrada;\n");
422   fprintf(arq,"   end if;\n");
423   fprintf(arq,"end process;\n");
424   fprintf(arq,"\n");
425   fprintf(arq,"end Behavioral;\n");
426   fclose(arq);
427
428   arq = fopen("TCGENERIC.vhd", "w");
429   fprintf(arq,"--faz a transformada Census de uma determinada máscara\n");
430   fprintf(arq,"library IEEE;\n");
431   fprintf(arq,"use IEEE.STD_LOGIC_1164.ALL;\n");
432   fprintf(arq,"use IEEE.STD_LOGIC_ARITH.ALL;\n");
433   fprintf(arq,"\n");
434   fprintf(arq,"entity transformadacensus is\n");
435   fprintf(arq,"    Port ( \n");
436   fprintf(arq,"           valor1 : in  std_logic_vector(%d downto 0);\n",((TotalJanela*8)-
      1));
437   fprintf(arq,"           clk : in STD_LOGIC;\n");
438   fprintf(arq,"           controle : in std_logic;\n");
439   fprintf(arq,"           ready : out std_logic;\n");
440   fprintf(arq,"           resultado : out  std_logic_vector(%d downto 0)\n",TotalJanela -
      2);
441   fprintf(arq,");\n");
442   fprintf(arq,"end transformadacensus;\n");
443   fprintf(arq,"\n");
444   fprintf(arq,"architecture Behavioral of transformadacensus is\n");
445   fprintf(arq,"\n");
446   fprintf(arq,"signal s_controle : std_logic_vector (%d downto 0):=(others=>'0');\n",
      TotalJanela - 1);
447   fprintf(arq,"signal v_saida : std_logic_vector(%d downto 0):=(others=>'0');\n",
      TotalJanela - 2);
448   fprintf(arq,"signal s_vref : std_logic_vector (7 downto 0);\n");
```

```
449   fprintf(arq,"\n");
450   fprintf(arq,"component comp\n");
451   fprintf(arq,"    Port ( ref : in  std_logic_vector(7 downto 0);\n");
452   fprintf(arq,"           valor : in  std_logic_vector(7 downto 0);\n");
453   fprintf(arq,"           controle : in std_logic;\n");
454   fprintf(arq,"           clk : in STD_LOGIC;\n");
455   fprintf(arq,"           s_controle : out STD_LOGIC;\n");
456   fprintf(arq,"           saida : out  std_logic);\n");
457   fprintf(arq,"end component;\n");
458   fprintf(arq,"\n");
459   fprintf(arq,"begin\n");
460   fprintf(arq,"s_vref <= valor1(%d downto %d);\n",TotalJanela*4+3,TotalJanela*4-4);
461   fprintf(arq,"ready <= s_controle(0);\n");
462   fprintf(arq,"gen_comp:\n");
463   fprintf(arq,"for n in 1 to %d generate\n",(TotalJanela-1)/2);
464   fprintf(arq,"comparador : comp port map \n");
465   fprintf(arq,"                       (ref => s_vref,\n");
466   fprintf(arq,"                        valor => valor1((8*n)-1 downto (8*n)-8),\n");
467   fprintf(arq,"                        controle => controle,\n");
468   fprintf(arq,"                        s_controle => s_controle(n-1),\n");
469   fprintf(arq,"                        clk => clk,\n");
470   fprintf(arq,"                        saida => v_saida(n-1));\n");
471   fprintf(arq,"end generate gen_comp;\n");
472   fprintf(arq,"gen_comp2:\n");
473   fprintf(arq,"for n in %d to %d generate\n",(TotalJanela-1)/2+2,TotalJanela);
474   fprintf(arq,"comparador : comp port map\n");
475   fprintf(arq,"                       (ref => s_vref,\n");
476   fprintf(arq,"                        valor => valor1((8*n)-1 downto (8*n)-8),\n");
477   fprintf(arq,"                        controle => controle,\n");
478   fprintf(arq,"                        s_controle => s_controle(n-1),\n");
479   fprintf(arq,"                        clk => clk,\n");
480   fprintf(arq,"                        saida => v_saida(n-2));\n");
481   fprintf(arq,"end generate gen_comp2;\n");
482   fprintf(arq,"resultado <= v_saida;\n");
483   fprintf(arq,"end Behavioral;\n");
484   fclose(arq);
485
486   arq = fopen("comp.vhd", "w");
487   fprintf(arq,"library IEEE;\n");
488   fprintf(arq,"use IEEE.STD_LOGIC_1164.ALL;\n");
489   fprintf(arq,"\n");
490   fprintf(arq,"entity comp is\n");
491   fprintf(arq,"    Port ( ref : in  std_logic_vector(7 downto 0);\n");
492   fprintf(arq,"           valor : in  std_logic_vector(7 downto 0);\n");
493   fprintf(arq,"             clk : in STD_LOGIC;\n");
494   fprintf(arq,"             controle : in std_logic;\n");
495   fprintf(arq,"             s_controle : out STD_LOGIC;\n");
496   fprintf(arq,"           saida : out  std_logic);\n");
497   fprintf(arq,"end comp;\n");
498   fprintf(arq,"\n");
499   fprintf(arq,"architecture Behavioral of comp is\n");
500   fprintf(arq,"\n");
501   fprintf(arq,"begin\n");
502   fprintf(arq,"process(clk,controle,ref,valor)\n");
```

```
503   fprintf(arq,"   begin\n");
504   fprintf(arq,"\n");
505   fprintf(arq,"   if rising_edge (clk) then\n");
506   fprintf(arq,"           if(controle = '1') then \n");
507   fprintf(arq,"               s_controle <= '1';\n");
508   fprintf(arq,"               if(valor > ref) then\n");
509   fprintf(arq,"                   saida <= '1';\n");
510   fprintf(arq,"               else\n");
511   fprintf(arq,"                   saida <= '0';\n");
512   fprintf(arq,"               end if;\n");
513   fprintf(arq,"           else \n");
514   fprintf(arq,"               s_controle <= '0';\n");
515   fprintf(arq,"               saida <='0';\n");
516   fprintf(arq,"           end if;\n");
517   fprintf(arq,"   end if;\n");
518   fprintf(arq,"end process;\n");
519   fprintf(arq,"end Behavioral;\n");
520   fclose(arq);
521
522   arq = fopen("DeslTransCensus.vhd", "w");
523   fprintf(arq,"library IEEE;\n");
524   fprintf(arq,"use IEEE.STD_LOGIC_1164.ALL;\n");
525   fprintf(arq,"use IEEE.STD_LOGIC_UNSIGNED.ALL;\n");
526   fprintf(arq,"\n");
527   fprintf(arq,"entity DTC is\n");
528   fprintf(arq,"    Port ( TCDin : in  std_logic_vector (%d downto 0);\n",TotalJanela-2);
529   fprintf(arq,"                   TCDout : out  std_logic_vector (%d downto 0);\n",(
      NUMCOLUNAS-t_mascara+1)*(TotalJanela-1)-1);
530   fprintf(arq,"                  DISPout : out  std_logic_vector (%d downto 0);\n",(
      NUMCOLUNAS-t_mascara+1)*(numero_bits_disp+1)-1);
531   fprintf(arq,"                   reset : in std_logic;\n");
532   fprintf(arq,"                   done : out std_logic;\n");
533   fprintf(arq,"                   start : in std_logic;\n");
534   fprintf(arq,"                   clk : in std_logic\n");
535   fprintf(arq,"                 );\n");
536   fprintf(arq,"end DTC;\n");
537   fprintf(arq,"\n");
538   fprintf(arq,"architecture Behavioral of DTC is\n");
539   fprintf(arq,"signal s_ligacao_TCD : std_logic_vector(%d downto 0):=(others => '1');\n",(
      NUMCOLUNAS-t_mascara+2)*(TotalJanela-1)-1);
540   fprintf(arq,"signal s_ligacao_disp : std_logic_vector(%d downto 0):=(others => '0');\n"
      ,(NUMCOLUNAS-t_mascara+2)*(numero_bits_disp+1)-1);
541   char disp_max_2[numero_bits_disp+1];
542   itoa(disparidade_maxima-1,disp_max_2,2);
543   fprintf(arq,"constant constante : std_logic_vector(%d downto 0):=\"%s\";\n",
      numero_bits_disp,disp_max_2);
544   fprintf(arq,"signal count : std_logic_vector(%d downto 0):=\"");
545   for(int i=0;i<numero_bits_disp;i++)
546       fprintf(arq,"0");
547   fprintf(arq,"1");
548   fprintf(arq,"\";\n",numero_bits_disp);
549   fprintf(arq,"signal s_reset: std_logic;\n");
550   fprintf(arq,"\n");
551   fprintf(arq,"component ADTC is\n");
```

```
552   fprintf(arq,"   port(   entradaTCD: in std_logic_vector(%d downto 0);\n",TotalJanela-2);
553   fprintf(arq,"               disp : in std_logic_vector(%d downto 0);\n",numero_bits_disp
      );
554   fprintf(arq,"               start: in std_logic;\n");
555   fprintf(arq,"               reset: in std_logic;\n");
556   fprintf(arq,"               clk: in std_logic;\n");
557   fprintf(arq,"               saidaTCD: out std_logic_vector(%d downto 0);\n",TotalJanela-
      2);
558   fprintf(arq,"               saidaDISP: out std_logic_vector(%d downto 0)\n",
      numero_bits_disp);
559   fprintf(arq,"               );\n");
560   fprintf(arq,"end component;\n");
561   fprintf(arq,"begin\n");
562   fprintf(arq,"s_ligacao_TCD(%d downto 0)<=TCDin;\n",TotalJanela-2);
563   fprintf(arq,"s_ligacao_disp(%d downto 0)<=\"",numero_bits_disp);
564   for(int j=0;j<=numero_bits_disp;j++)
565           fprintf(arq,"0");
566   fprintf(arq,"\";\n");
567   fprintf(arq,"\n");
568   fprintf(arq,"TCDout <= s_ligacao_TCD(%d downto %d);\n",(NUMCOLUNAS-t_mascara+2)*(
      TotalJanela-1)-1,TotalJanela-1);
569   fprintf(arq,"DISPout <= s_ligacao_disp(%d downto %d);\n",(NUMCOLUNAS-t_mascara+2)*(
      numero_bits_disp+1)-1,numero_bits_disp+1);
570   fprintf(arq,"\n");
571   fprintf(arq,"gen_adp:\n");
572   fprintf(arq,"for n in 2 to %d generate--5 linhas\n",disparidade_maxima);
573   fprintf(arq,"comp : ADTC port map \n");
574   fprintf(arq,"               (entradaTCD=>s_ligacao_TCD((n*%d-1) downto (n-1)*%d),\n",
      TotalJanela-1,TotalJanela-1);
575   fprintf(arq,"               disp =>s_ligacao_disp((n*%d-1) downto (n-1)*%d),\n",
      numero_bits_disp+1,numero_bits_disp+1);
576   fprintf(arq,"               start=>reset,\n");
577   fprintf(arq,"               reset=>s_reset,\n");
578   fprintf(arq,"               clk=>clk,\n");
579   fprintf(arq,"               saidaTCD=>s_ligacao_TCD(((n+1)*%d-1) downto n*%d),\n",
      TotalJanela-1,TotalJanela-1);
580   fprintf(arq,"               saidaDISP=>s_ligacao_disp(((n+1)*%d-1) downto n*%d)\n",
      numero_bits_disp+1,numero_bits_disp+1);
581   fprintf(arq,"               );\n");
582   fprintf(arq,"end generate gen_adp;\n");
583   fprintf(arq,"comp : ADTC port map \n");
584   fprintf(arq,"               (entradaTCD=>s_ligacao_TCD(%d downto 0),\n",TotalJanela-2);
585   fprintf(arq,"               disp =>s_ligacao_disp(%d downto 0),\n",numero_bits_disp);
586   fprintf(arq,"               start=>reset,\n");
587   fprintf(arq,"               reset=>'0',\n");
588   fprintf(arq,"               clk=>clk,\n");
589   fprintf(arq,"               saidaTCD=>s_ligacao_TCD(%d downto %d),\n",(TotalJanela-1)*2-
      1,TotalJanela-1);
590   fprintf(arq,"               saidaDISP=>s_ligacao_disp(%d downto %d)\n",(numero_bits_disp
      +1)*2-1,numero_bits_disp+1);
591   fprintf(arq,"               );\n");
592   fprintf(arq,"\n");
593   fprintf(arq,"\n");
594   fprintf(arq,"process(clk,reset,start)\n");
```

```
595   fprintf(arq,"begin\n");
596   fprintf(arq,"   if reset = '0' or start = '0' then\n");
597   fprintf(arq,"       count<=\"");
598   for(int i=0;i<numero_bits_disp;i++)
599       fprintf(arq,"0");
600   fprintf(arq,"1");
601   fprintf(arq,"\";\n");
602   fprintf(arq,"       s_reset <= '1';\n");
603   fprintf(arq,"       done<='0';\n");
604   fprintf(arq,"\n");
605   fprintf(arq,"   elsif rising_edge(clk) then\n");
606   fprintf(arq,"           done<='1';\n");
607   fprintf(arq,"           if (count>constante) then\n");
608   fprintf(arq,"               s_reset <= '1';\n");
609   fprintf(arq,"               count<= \"");
610   for(int i=0;i<numero_bits_disp;i++)
611       fprintf(arq,"0");
612   fprintf(arq,"1");
613   fprintf(arq,"\";\n");
614   fprintf(arq,"           else\n");
615   fprintf(arq,"               s_reset <= '0';\n");
616   fprintf(arq,"               count <= count +\"");
617   for(int i=0;i<numero_bits_disp;i++)
618       fprintf(arq,"0");
619   fprintf(arq,"1");
620   fprintf(arq,"\";\n");
621   fprintf(arq,"           \n");
622   fprintf(arq,"       end if;\n");
623   fprintf(arq,"   end if;\n");
624   fprintf(arq,"end process;\n");
625   fprintf(arq,"\n");
626   fprintf(arq,"\n");
627   fprintf(arq,"\n");
628   fprintf(arq,"end Behavioral;\n");
629   fprintf(arq,"\n");
630   fclose(arq);
631
632   arq = fopen("ADTC.vhd", "w");
633   fprintf(arq,"library IEEE;\n");
634   fprintf(arq,"use IEEE.STD_LOGIC_1164.ALL;\n");
635   fprintf(arq,"use IEEE.STD_LOGIC_UNSIGNED.ALL;\n");
636   fprintf(arq,"\n");
637   fprintf(arq,"entity ADTC is\n");
638   fprintf(arq,"   port(   entradaTCD: in std_logic_vector(%d downto 0);\n",TotalJanela-2);
639   fprintf(arq,"               disp : in std_logic_vector(%d downto 0);\n",numero_bits_disp
      );
640   fprintf(arq,"               start: in std_logic;\n");
641   fprintf(arq,"               reset: in std_logic;\n");
642   fprintf(arq,"               clk: in std_logic;\n");
643   fprintf(arq,"               saidaTCD: out std_logic_vector(%d downto 0);\n",TotalJanela-
      2);
644   fprintf(arq,"               saidaDISP: out std_logic_vector(%d downto 0)\n",
      numero_bits_disp);
645   fprintf(arq,"               );\n");
```

```
646   fprintf(arq,"end ADTC;\n");
647   fprintf(arq,"\n");
648   fprintf(arq,"architecture Behavioral of ADTC is\n");
649   fprintf(arq,"begin\n");
650   fprintf(arq,"process(clk,reset,start)\n");
651   fprintf(arq,"begin\n");
652   fprintf(arq,"   if (start = '0') then \n");
653   fprintf(arq,"       saidaTCD<=(others=>'0');\n");
654   fprintf(arq,"       saidaDISP<=(others=>'0');\n");
655   fprintf(arq,"   elsif rising_edge(clk) then\n");
656   fprintf(arq,"   saidaTCD <= entradaTCD;\n");
657   fprintf(arq,"       if(reset='0') then\n");
658   fprintf(arq,"           saidaDISP <= disp+'1';\n");
659   fprintf(arq,"       else\n");
660   fprintf(arq,"           saidaDISP<=(others=>'0');\n");
661   fprintf(arq,"       end if;\n");
662   fprintf(arq,"   end if;\n");
663   fprintf(arq,"end process;\n");
664   fprintf(arq,"\n");
665   fprintf(arq,"end Behavioral;\n");
666   fprintf(arq,"\n");
667   fclose(arq);
668
669   arq = fopen("Atraso_TCE.vhd", "w");
670   fprintf(arq,"library IEEE;\n");
671   fprintf(arq,"use IEEE.STD_LOGIC_1164.ALL;\n");
672   fprintf(arq,"\n");
673   fprintf(arq,"entity Atraso_TCE is\n");
674   fprintf(arq,"Port (         entrada : in  std_logic_vector(%d downto 0);\n",TotalJanela-
      2);
675   fprintf(arq,"               reset: in std_logic;\n");
676   fprintf(arq,"               clk: in std_logic;\n");
677   fprintf(arq,"               saida : out std_logic_vector(%d downto 0));\n",TotalJanela-2
      );
678   fprintf(arq,"end Atraso_TCE;\n");
679   fprintf(arq,"\n");
680   fprintf(arq,"architecture Behavioral of Atraso_TCE is\n");
681   fprintf(arq,"signal s_saida : std_logic_vector(%d downto 0):=(others => '0');\n",
      TotalJanela-2);
682   fprintf(arq,"begin\n");
683   fprintf(arq,"saida<=s_saida;\n");
684   fprintf(arq,"process(clk,reset)\n");
685   fprintf(arq,"begin\n");
686   fprintf(arq,"   if (reset = '0') then \n");
687   fprintf(arq,"       s_saida<=(others=>'0');\n");
688   fprintf(arq,"   elsif rising_edge(clk) then\n");
689   fprintf(arq,"       s_saida <= entrada;\n");
690   fprintf(arq,"   end if;\n");
691   fprintf(arq,"end process;\n");
692   fprintf(arq,"\n");
693   fprintf(arq,"end Behavioral;\n");
694   fprintf(arq,"\n");
695   fclose(arq);
696   int i,j=1,s1;
```

```
697   double x;
698   x=int(log2(TotalJanela-1));
699   s1=(TotalJanela-1)/4;
700   //quanidade de processos
701   int a,b,count=1,aux,aux2;
702   a=(TotalJanela-1)/4;
703   b=(TotalJanela-1)%4;
704   if(b!=0)a++;//TotalJanelaero de sinais resultado no primeiro processo.
705   while(a!=1){
706       b=a%2;
707       a=a/2;
708       if(b!=0)a++;
709       count++;
710       }
711
712   arq = fopen("DH.vhd", "w");
713   fprintf(arq,"library IEEE;\n");
714   fprintf(arq,"use IEEE.STD_LOGIC_1164.ALL;\n");
715   fprintf(arq,"use IEEE.std_logic_arith.all;\n");
716   fprintf(arq,"USE ieee.std_logic_unsigned.all;\n");
717   fprintf(arq,"entity DH is\n");
718   fprintf(arq,"   port(vetor1: in std_logic_vector(%d downto 0);\n",TotalJanela-2);
719   fprintf(arq,"        vetor2: in std_logic_vector(%d downto 0);\n",TotalJanela-2);
720
721   fprintf(arq,"        DISPin: in std_logic_vector(%d downto 0);\n",numero_bits_disp);
722   fprintf(arq,"        DISPout: out std_logic_vector(%d downto 0);\n",numero_bits_disp);
723
724   fprintf(arq,"        start: in std_logic;\n");
725   fprintf(arq,"        done: out std_logic;\n");
726   fprintf(arq,"        clk: in std_logic;\n");
727   fprintf(arq,"        resultado: out std_logic_vector(%2.0f downto 0)\n",x);
728   fprintf(arq,"        );\n");
729   fprintf(arq,"end DH;\n");
730   fprintf(arq,"architecture Behavioral of DH is\n");
731   int y=count-1;
732   int k=int(log2(y));
733   char c[k+1];
734   itoa(y,c,2);
735   //fprintf(arq,"constant constante : unsigned(%d downto 0):=\"%s\";\n",k,c);
736   //fprintf(arq,"signal count : unsigned(%d downto 0):=(others => '0');\n\n",k);
737   fprintf(arq,"signal s_xor: std_logic_vector (%d downto 0):= (others => '0');\n\n",
      TotalJanela-2);
738   for(i=1;i<TotalJanela;i++)
739       fprintf(arq,"signal s_aux%d: std_logic_vector (%2.0f downto 0):= (others => '0');\n"
          ,i,x);
740   fprintf(arq,"\n");
741   while(s1>1){
742       for(i=0;i<s1;i++)
743           fprintf(arq,"signal s%d_%d: std_logic_vector (%2.0f downto 0):= (others =>
              '0');\n",j,i+1,x);
744       fprintf(arq,"signal s_disp%d: std_logic_vector(%d downto 0);\n",j,numero_bits_disp);
745       fprintf(arq,"\n");
746       b=s1%2;
747       s1/=2;
```

```
748         if(b!=0)s1++;
749         j++;
750     }
751     fprintf(arq,"signal s%d_%d: std_logic_vector (%2.0f downto 0):= (others => '0');\n\n",j,
        1,x);
752     fprintf(arq,"signal s_disp%d: std_logic_vector(%d downto 0);\n",j,numero_bits_disp);
753     fprintf(arq,"signal sdone: std_logic_vector (%d downto 0):= (others => '0');\n\n",j-1);
754     for(i=0;i<s1;i++)
755     fprintf(arq,"begin\n");
756     fprintf(arq,"DISPout<=s_disp%d;\n",count);
757     fprintf(arq,"resultado<=s%d_%d;\n",j,1);
758     fprintf(arq,"s_xor <= vetor1 xor vetor2;\n");
759     fprintf(arq,"done <= sdone(%d);\n",count-1);
760     for(int n=0;n<TotalJanela-1;n++){
761         fprintf(arq,"s_aux%d <= \"",n+1);
762         for(i=0;i<((int)x);i++)
763             fprintf(arq,"0");
764         fprintf(arq,"\"&s_xor(%d);\n",n);
765         }
766     fprintf(arq,"\nprocess(clk,start)\n");
767     fprintf(arq,"\tbegin\n");
768     fprintf(arq,"\t\tif (start = '0') then \n");
769     for(int n=0;n<(TotalJanela-1)/4;n++)
770         fprintf(arq,"\t\t\ts%d_%d<=(others=>'0');\n",1,n+1);
771     fprintf(arq,"\t\t\ts_disp1<=(others=>'0');\n");
772     fprintf(arq,"\t\t\tsdone(0)<='0';\n");
773     fprintf(arq,"\t\telsif rising_edge(clk) then\n");
774     fprintf(arq,"\t\t\ts_disp1<=DISPin;\n");
775     fprintf(arq,"\t\t\tsdone(0)<='1';\n");
776     for(j=0;j<(TotalJanela-1)/4;j++){
777         fprintf(arq,"\t\t\ts%d_%d <= ",1,j+1);
778         for(i=1;i<5;i++){
779             fprintf(arq,"s_aux%d",j*4+i);
780             if(i!=4)fprintf(arq," + ");
781             }
782         fprintf(arq,";\n");
783         }
784     fprintf(arq,"\t\tend if;\n");
785     fprintf(arq,"\tend process;\n");
786     aux=(TotalJanela-1)/8;
787     if(((TotalJanela-1)%8)>0)aux++;
788     aux2=0;
789     for(b=1;b<count;b++){
790         fprintf(arq,"\nprocess(clk,start)\n");
791         fprintf(arq,"\tbegin\n");
792         fprintf(arq,"\t\tif (start = '0') then \n");
793         fprintf(arq,"\t\t\tsdone(%d)<='0';\n",b);
794         for(j=0;j<(aux);j++)
795             fprintf(arq,"\t\t\ts%d_%d<=(others=>'0');\n",b+1,j+1);
796         fprintf(arq,"\t\t\ts_disp%d<=(others=>'0');\n",b+1);
797         fprintf(arq,"\t\telsif rising_edge(clk) then\n");
798         fprintf(arq,"\t\t\ts_disp%d<=s_disp%d;\n",b+1,b);
799         fprintf(arq,"\t\t\tsdone(%d)<=sdone(%d);\n",b,b-1);
800         for(j=0;j<(aux);j++){
```

```
801             fprintf(arq,"\t\t\ts%d_%d <= ",b+1,j+1);
802             for(i=1;i<3;i++){
803                 if(i==2 && aux2%2!=0 && j+1==aux )
804                     break;
805                 if(i!=1)
806                     fprintf(arq," + ");
807                 fprintf(arq,"s%d_%d",b,j*2+i);
808                 }
809             fprintf(arq,";\n");
810         }
811         fprintf(arq,"\t\tend if;\n");
812         fprintf(arq,"\tend process;\n");
813         aux2=aux;
814         aux/=2;
815         if(aux2%2!=0)
816             aux++;
817     }
818     /*
819     fprintf(arq,"process(clk,start)\n");
820     fprintf(arq,"begin\n");
821     fprintf(arq,"   if start = '1' then\n");
822     fprintf(arq,"       count<=(others => '0');\n");
823     fprintf(arq,"       ready<='0';\n");
824     fprintf(arq,"   elsif rising_edge(clk) then\n");
825     fprintf(arq,"       if (count=constante) then\n");
826     fprintf(arq,"           ready<='1';\n");
827     fprintf(arq,"       else\n");
828     fprintf(arq,"           ready<='0';\n");
829     fprintf(arq,"           count <= count +'1';\n");
830     fprintf(arq,"       end if;\n");
831     fprintf(arq,"   end if;\n");
832     fprintf(arq,"end process;\n");
833     fprintf(arq,"\n");
834     fprintf(arq,"resultado <= s%d_%d;\n",count,aux);
835     fprintf(arq,"\n");*/
836     fprintf(arq,"\nend Behavioral;");
837     fclose(arq);
838
839     arq = fopen("compDISP.vhd", "w");
840     fprintf(arq,"library IEEE;\n");
841     fprintf(arq,"use IEEE.STD_LOGIC_1164.ALL;\n");
842     fprintf(arq,"\n");
843     fprintf(arq,"\n");
844     fprintf(arq,"entity teste_comparacao is\n");
845     fprintf(arq,"    Port (DistHamming1: in std_logic_vector (%d downto 0);\n",
        numero_bits_dh-1);
846     fprintf(arq,"          DistHamming2: in std_logic_vector (%d downto 0);\n",
        numero_bits_dh-1);
847     fprintf(arq,"          disparidade1: in std_logic_vector(%d downto 0);\n",
        numero_bits_disp);
848     fprintf(arq,"          disparidade2: in std_logic_vector(%d downto 0);\n",
        numero_bits_disp);
849     fprintf(arq,"          s_disparidade: out std_logic_vector(%d downto 0);\n",
        numero_bits_disp);
```

```
850   fprintf(arq,"          clk : in std_logic;\n");
851   fprintf(arq,"          s_DistHamming: out std_logic_vector ( %d downto 0)\n",
      numero_bits_dh-1);
852   fprintf(arq,"          );\n");
853   fprintf(arq,"end teste_comparacao;\n");
854   fprintf(arq,"");
855   fprintf(arq,"architecture Behavioral of teste_comparacao is\n");
856   fprintf(arq,"\n");
857   fprintf(arq,"begin\n");
858   fprintf(arq,"process(clk)\n");
859   fprintf(arq,"   begin\n");
860   fprintf(arq,"   if rising_edge(clk) then\n");
861   fprintf(arq,"       if(DistHamming1 <= DistHamming2 or disparidade2=\"");
862   for(int i=0;i<=numero_bits_disp;i++)
863           fprintf(arq,"0");
864   fprintf(arq,"\") then \n");
865   fprintf(arq,"           s_DistHamming<=DistHamming1;\n");
866   fprintf(arq,"           s_disparidade<=disparidade1;\n");
867   fprintf(arq,"       else\n");
868   fprintf(arq,"           s_DistHamming <=DistHamming2;\n");
869   fprintf(arq,"           s_disparidade<=disparidade2;\n");
870   fprintf(arq,"       end if;\n");
871   fprintf(arq,"   end if;\n");
872   fprintf(arq,"end process;\n");
873   fprintf(arq,"\n");
874   fprintf(arq,"end Behavioral;\n");
875   fprintf(arq,"\n");
876   fclose(arq);
877
878   arq = fopen("TOTALcompDISP.vhd", "w");
879
880   fprintf(arq,"library IEEE;\n");
881   fprintf(arq,"use IEEE.STD_LOGIC_1164.ALL;\n");
882   fprintf(arq,"\n");
883   fprintf(arq,"entity TOTALcompDISP is\n");
884   fprintf(arq,"                   port(\n");
885   fprintf(arq,"                   resultadoDH: in std_logic_vector(%d downto 0);\n",
      numero_bits_dh*(NUMCOLUNAS-t_mascara+1)-1);
886   fprintf(arq,"                   DISPoutDH: in std_logic_vector(%d downto 0);\n",(
      numero_bits_disp+1)*(NUMCOLUNAS-t_mascara+1)-1);
887   fprintf(arq,"                   DISPfinal: out std_logic_vector(%d downto 0);\n",
      numero_bits_disp);
888   fprintf(arq,"                   clk: in std_logic\n");
889   fprintf(arq,"                   );\n");
890   fprintf(arq,"end TOTALcompDISP;\n");
891   fprintf(arq,"\n");
892   fprintf(arq,"architecture Behavioral of TOTALcompDISP is\n");
893   fprintf(arq,"\n");
894   fprintf(arq,"component teste_comparacao\n");
895   fprintf(arq,"    Port (DistHamming1: in std_logic_vector (%d downto 0);\n",
      numero_bits_dh-1);
896   fprintf(arq,"          DistHamming2: in std_logic_vector (%d downto 0);\n",
      numero_bits_dh-1);
897   fprintf(arq,"          disparidade1: in std_logic_vector(%d downto 0);\n",
```

```
    numero_bits_disp);
898 fprintf(arq,"          disparidade2: in std_logic_vector(%d downto 0);\n",
    numero_bits_disp);
899 fprintf(arq,"          s_disparidade: out std_logic_vector(%d downto 0);\n",
    numero_bits_disp);
900 fprintf(arq,"          clk : in std_logic;\n");
901 fprintf(arq,"          s_DistHamming: out std_logic_vector ( %d downto 0)\n",
    numero_bits_dh-1);
902 fprintf(arq,"          );\n");
903 fprintf(arq,"end component;\n");
904 fprintf(arq,"\n");
905 fprintf(arq,"\n");
906 float teste=log2(disparidade_maxima);
907 int num_sinais=(int)teste;
908 if(teste-num_sinais!=0)num_sinais++;
909 aux=NUMCOLUNAS-t_mascara+1;
910 for(int i=0;i<num_sinais;i++){
911      if(aux%2!=0)aux++;
912          fprintf(arq,"signal s_ligacao_dh_%d: std_logic_vector(%d downto 0):=(others =>
             '0');\n",i+1,(numero_bits_dh)*aux/2-1);
913          fprintf(arq,"signal s_ligacao_disp_%d: std_logic_vector(%d downto 0):=(others
             => '0');\n",i+1,(numero_bits_disp+1)*aux/2-1);
914      aux/=2;
915      fprintf(arq,"\n");
916 }
917 fprintf(arq,"\n");
918 if(NUMCOLUNAS%2==0){
919     fprintf(arq,"signal sresultadoDH : std_logic_vector(%d downto 0):=(others =>
        '0');\n",numero_bits_dh*(NUMCOLUNAS-t_mascara+1)-1);
920     fprintf(arq,"signal sDISPoutDH : std_logic_vector(%d downto 0):=(others => '0');\n"
        ,(numero_bits_disp+1)*(NUMCOLUNAS-t_mascara+1)-1);
921     }
922 else{
923      fprintf(arq,"signal sresultadoDH : std_logic_vector(%d downto 0):=(others =>
        '0');\n",(numero_bits_dh)*(NUMCOLUNAS-t_mascara+2)-1);
924      fprintf(arq,"signal sDISPoutDH : std_logic_vector(%d downto 0):=(others => '0');\n"
        ,(numero_bits_disp+1)*(NUMCOLUNAS-t_mascara+2)-1);
925      }
926 fprintf(arq,"begin\n");
927 fprintf(arq,"DISPfinal <= s_ligacao_disp_%d;\n",num_sinais);
928 fprintf(arq,"\n-------------------------------------------------------------------------\n\n");
929 fprintf(arq,"sresultadoDH(%d downto 0) <= resultadoDH;\n",numero_bits_dh*(NUMCOLUNAS-
    t_mascara+1)-1);
930 fprintf(arq,"sDISPoutDH(%d downto 0) <= DISPoutDH;\n",(numero_bits_disp+1)*(NUMCOLUNAS-
    t_mascara+1)-1);
931 fprintf(arq,"\n");
932 fprintf(arq,"\n");
933 fprintf(arq,"gen_comp1:\n");
934 int numero=(NUMCOLUNAS-t_mascara+1);
935 int numeroX=numero/2;
936 if(numero%2!=0)numeroX++;
937
938 fprintf(arq,"for n in 1 to %d generate\n",numeroX);
939 fprintf(arq,"compCOMPdh : teste_comparacao port map \n");
```

```
940   fprintf(arq,"                        (DistHamming1 => sresultadoDH(%d+%d*(n-1) downto
      %d*(n-1)),\n",numero_bits_dh-1,numero_bits_dh*2,numero_bits_dh*2);
941   fprintf(arq,"                        DistHamming2 => sresultadoDH(%d+%d*(n-1) downto
      %d+%d*(n-1)),\n",2*numero_bits_dh-1,numero_bits_dh*2,numero_bits_dh,numero_bits_dh*2);
942   fprintf(arq,"                        disparidade1 => sDISPoutDH(%d+%d*(n-1) downto
      %d*(n-1)),\n",numero_bits_disp,(numero_bits_disp+1)*2,(numero_bits_disp+1)*2);
943   fprintf(arq,"                        disparidade2 => sDISPoutDH(%d+%d*(n-1) downto
      %d+%d*(n-1)),\n",(numero_bits_disp+1)*2-1,(numero_bits_disp+1)*2,numero_bits_disp+1,(
      numero_bits_disp+1)*2);
944   fprintf(arq,"                        s_disparidade => s_ligacao_disp_1(%d+%d*(n-1)
      downto %d*(n-1)),\n",numero_bits_disp,(numero_bits_disp+1),(numero_bits_disp+1));
945   fprintf(arq,"                        clk => clk,\n");
946   fprintf(arq,"                        s_DistHamming =>s_ligacao_dh_1(%d+%d*(n-1) downto
      %d*(n-1)));\n",numero_bits_dh-1,numero_bits_dh,numero_bits_dh);
947   fprintf(arq,"end generate gen_comp1;\n");
948   fprintf(arq,"\n");
949   //printf("\n%d",numero);system("PAUSE");
950   numero=numeroX;
951   numeroX= numero;
952   for(int j=1;numero>1;j++){
953   numero=numeroX;
954   numeroX/=2;
955
956   if((numero%2)!=0 && numero!=1)numeroX++;
957   if(numeroX!=0){
958   fprintf(arq,"gen_comp%d:\n",j+1);
959   fprintf(arq,"for n in 1 to %d generate\n",numeroX);
960   fprintf(arq,"compCOMPdh : teste_comparacao port map \n");
961   fprintf(arq,"                        (DistHamming1 => s_ligacao_dh_%d(%d+%d*(n-1) downto
      %d*(n-1)),\n",j,numero_bits_dh-1,numero_bits_dh*2,numero_bits_dh*2);
962   fprintf(arq,"                        DistHamming2 => s_ligacao_dh_%d(%d+%d*(n-1) downto
      %d+%d*(n-1)),\n",j,2*numero_bits_dh-1,numero_bits_dh*2,numero_bits_dh,numero_bits_dh*2);
963   fprintf(arq,"                        disparidade1 => s_ligacao_disp_%d(%d+%d*(n-1)
      downto %d*(n-1)),\n",j,numero_bits_disp,(numero_bits_disp+1)*2,(numero_bits_disp+1)*2);
964   fprintf(arq,"                        disparidade2 => s_ligacao_disp_%d(%d+%d*(n-1)
      downto %d+%d*(n-1)),\n",j,(numero_bits_disp+1)*2-1,(numero_bits_disp+1)*2,
      numero_bits_disp+1,(numero_bits_disp+1)*2);
965   if(numero!=1){
966                fprintf(arq,"                        s_disparidade =>
                   s_ligacao_disp_%d(%d+%d*(n-1) downto %d*(n-1)),\n",j+1,numero_bits_disp,
                   numero_bits_disp+1,numero_bits_disp+1);
967                fprintf(arq,"                        clk => clk,\n");
968                fprintf(arq,"                        s_DistHamming
                   =>s_ligacao_dh_%d(%d+%d*(n-1) downto %d*(n-1)));\n",j+1,numero_bits_dh-1,
                   numero_bits_dh,numero_bits_dh);
969                }
970   else{
971        fprintf(arq,"                        s_disparidade =>s_ligacao_disp_%d,\n",j+1);
972        fprintf(arq,"                        clk => clk,\n");
973        fprintf(arq,"                        s_DistHamming =>s_ligacao_dh_%d);\n",j+1);
974        }
975   fprintf(arq,"end generate gen_comp%d;\n",j+1);
976   fprintf(arq,"\n");
977   }
```

```
978   }
979   fprintf(arq,"end Behavioral;\n");
980
981   fclose(arq);
982   }//main
983
```

ANEXO C – *Software* para converter resultado do *hardware* em imagem

```c
#include <stdio.h>
#include <stdlib.h>
#define TID 46//tamanho da imagem quadrada 46x46
    FILE *arq;
    FILE *arq2;
int main(){
    int a,count=0,result=0;
    arq=fopen("resultados_disp.txt","r");
    arq2=fopen("disp.txt","w");
    while(count<TID*TID){
        result=0;
        count++;
        fscanf(arq,"%d",&a);
        if(a>=10000000){
            result=128;
            a-=10000000;
           }
        if(a>=1000000){
            result+=64;
            a-=1000000;
            }
        if(a>=100000){
            result+=32;
            a-=100000;
            }
        if(a>=10000){
            result+=16;
            a-=10000;
            }
        if(a>=1000){
            result+=8;
            a-=1000;
            }
        if(a>=100){
            result+=4;
            a-=100;
            }
        if(a>=10){
            result+=2;
            a-=10;
            }
        if(a>=1)
            result+=1;
        printf("%d\n",count);
        if((count-1)%TID==0)
            fprintf(arq2,"\n");
        fprintf(arq2,"%d\t",result);
        }
    fclose(arq);
    fclose(arq2);
}

```